Universidad
Carlos III de Madrid

Departamento de Teoría de la Señal y Comunicaciones

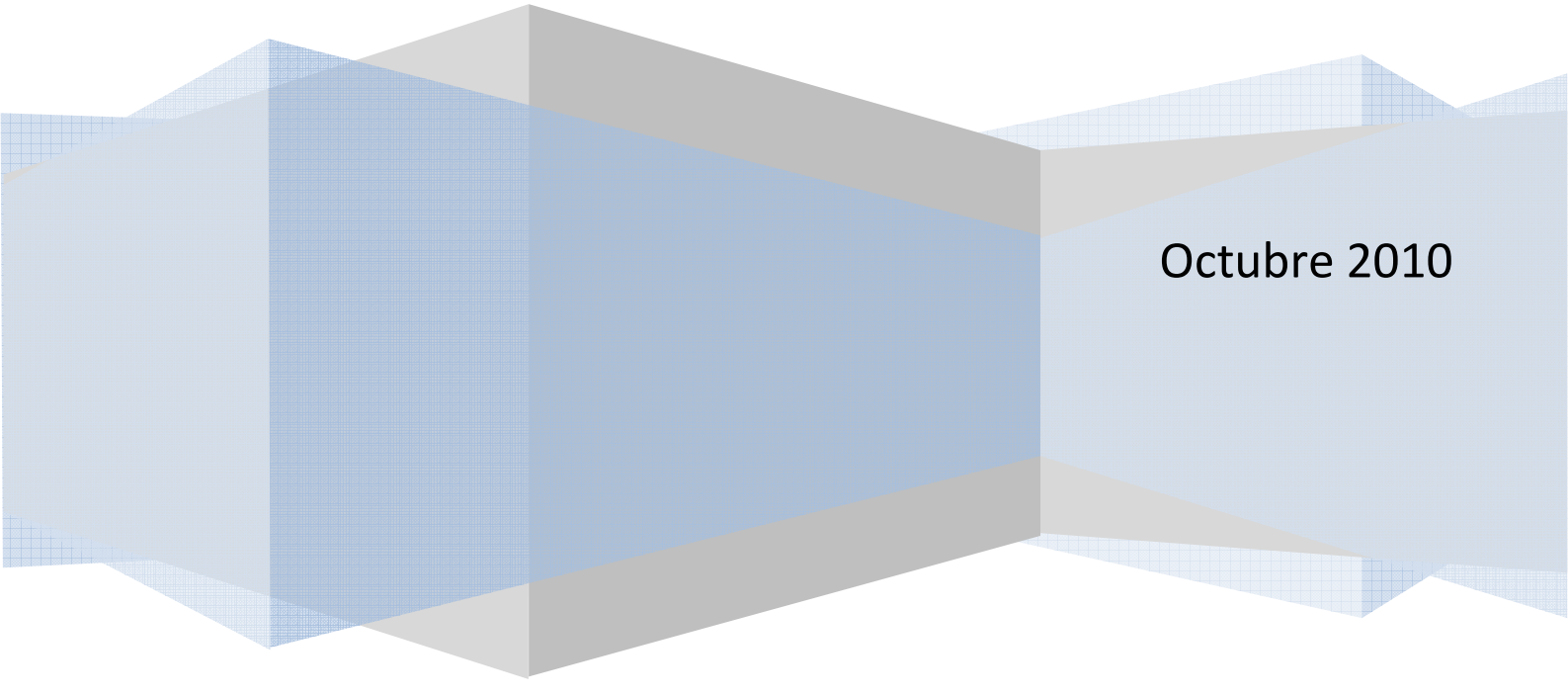
Proyecto final de carrera

Descripción Hardware de Algortimos de estimación de canal y sincronización tiempo-frecuencia para un sistema 2x2 MIMO-OFDM

Autor: Tomás Alemany Sánchez

Tutores: Dr. Víctor Pedro Gil Jiménez

Dr. Enrique San Millán



Octubre 2010

El Tribunal

Presidente: **Ana García Armada**

Vocal: **M. Julia Fernández-Getino**

Secretario: **Mario García**

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día **28 de Octubre de 2010** en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de **Matrícula de Honor (MH)**

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

... a mis tutores, mi familia y mi novia Silvia

Resumen

Palabras clave: OFDM, MIMO, FPGA, estimación de canal y sincronización tiempo-frecuencia

La sociedad actual, cada vez más, demanda banda ancha y movilidad, por lo que se requieren cada vez más, sistemas de comunicaciones que cumplan estas dos premisas. Así, hoy en día, existe un gran interés por las redes de comunicaciones inalámbricas de alta capacidad.

El empleo de la modulación "*Orthogonal Frecuency Division Multiplexing*" (OFDM) combinada con técnicas multi-antena "*Multiple-Input Multiple-Output*" (MIMO) es uno de los métodos más atractivos para implementar sistemas de comunicaciones inalámbricos de alta capacidad. La modulación ODFM presenta una gran robustez ante canales selectivos en frecuencia, ya que divide el espectro de frecuencias en canales que presentan un desvanecimiento prácticamente plano, posibilitando así una sencilla igualación de los mismos. Por otra parte, las técnicas MIMO proporcionan diversidad y permiten un aumentar la capacidad del sistema de comunicación inalámbrico. Los sistemas que utilizan estas técnicas son conocidos como sistemas MIMO-OFDM.

Una FPGA (*Field Programmable Gate Array*) es un dispositivo semiconductor constituido por una serie de bloques lógicos programables (CLBs, *Configurable Logic Blocs*) que se interconectan a través de una matriz de interconexión programable electrónicamente. En los últimos años, las plataformas de desarrollo basadas en FPGAs han evolucionado considerablemente y la aparición de nuevas herramientas de diseño hardware, ha hecho posible que se pueda emplear este tipo de dispositivos para aplicaciones que antiguamente se destinaban a los ASIC (*Circuito integrado para aplicaciones especificas*)

El presente proyecto trata de completar un sistema de comunicaciones basado en los conceptos descritos en los párrafos anteriores, es decir, sistemas MIMO-OFDM y plataformas de desarrollo basadas en FPGAs. Así, el fin del presente proyecto es diseñar los módulos de estimación de canal y sincronización tiempo-frecuencia de un sistema de comunicaciones basado en el estándar IEEE 802.16d-2004 (*Wimax fijo*) utilizando MIMO 2x2, modulación OFDM y BPSK (*Binary Phase Shift Keying*) para la modulación de datos. El diseño de los módulos de estimación de canal y sincronización tiempo-frecuencia han sido

realizados mediante una plataforma de desarrollo sobre FPGA basada en Simulink®.

Abstract

Keywords: OFDM, MIMO, FPGA, Channel estimation and Time-frequency synchronization

Current society, nowadays, demand more broadband services and mobility, then, telecommunication systems which achieve these requirements, are required.

The Orthogonal Frequency Division Multiplexing (OFDM) modulation combined with *Multiple-Input Multiple-Output* (MIMO) techniques are one of the most interesting methods to provide broadband wireless telecommunication systems. OFDM modulation provides immunity in frequency selective fading channels. On the other hand, MIMO techniques provide diversity and allow increasing the capability in the wireless telecommunication systems.

A Field Programmable Gate Array (FPGA) is a semiconductor device composed of Configurable Logic Blocs (CLBs) which are interconnected by means of a programmable interconnection matrix. In the last years, developments platforms based on FPGAs have considerably evolved, and the appearance of new Hardware design tools, has made possible the use of this kind of devices in applications which were planned for ASICs (Application-Specific Integrated Circuit).

The main goal of this project is to complete a telecommunication system based on the concepts described in the previous paragraphs: MIMO-OFDM systems and developments platforms based on FPGAs. Then, the goal of this project is to design the channel estimation and the time-frequency synchronization modules of a telecommunication system based on IEEE 802.16d-2004 (Fixed Wimax) using MIMO 2x2, OFDM modulation and BPSK (Binary Phase Shift Keying) for data modulation. The design of the channel estimation and time-frequency modules have been made by means of a development platform on FPGA based in Simulink®.

Índice general

Índice de figuras

Índice de Tablas

Glosario

ASIC	Application Specific Integrated Circuit (Circuito Integrado para Aplicaciones Específicas)
BPSK	Binary Phase Shift Keying (Modulación Binaria por Salto de Fase)
BS	Base Station (Estación Base)
CLB	Configurable Logic Block (Bloque Lógico Configurable)
CP	Cyclic Prefix (Prefijo Cíclico)
cPCI	Compact Periferical Component Interconect (Interconexión de Componentes Periféricos Compactos)
DCM	Digital Clock Manager (Gestor de Reloj Digital)
DFT	Discrete Fourier Transform (Transformada Discreta de Fourier)
DL	Downlink (Enlace Descendente)
DSP	Digital Signal Processor (Procesador Digital de Señales)
FFT	Fast Fouirer Transform (Transformada Rápida de Fourier)
FPGA	Field Programmable Gate Array (Matriz de Puertas Programables)
HDL	Hardware Description Language (Lenguaje de Descripción de Soportes Físicos)
IDFT	Inverse Discrete Fourier Transform (Trasformada Inversa Discreta de Fourier)
IFFT	Inverse Fast Fouirer Transform (Transformada Rápida Inversa de Fourier)
IOB	Input Output Banks (Bancos de Entrada Salida)
LMMSE	Linear Minimum-Mean Squared Error (Mímino Error Cuadratico medio)
LOS	Line Of Sigth (Línea de Visión Directa)

LS	Least Square (Mínimos Cuadrados)
LUT	Look-Up Table (Tabla de Consulta)
MSE	Mean Squared Error (Error Cuadrático Medio)
MIMO	Multiple-Input Multiple-Output (Multiples Entradas Multiples Salidas)
ML	Maximun Likelihood (Máxima Verosimilitud)
NLOS	Non Line of Sight (Sin Línea de Visión Directa)
OFDM	Orthogonal Frequency Division Multiplexing (Multiplexado por División en Frecuencias Ortogonales)
QAM	Quadrature Amplitude Modulation (Modulación en Amplitud por Cuadratura)
QoS	Quality of Service (Calidad de Servicio)
QPSK	Quadrature Phase Shift Keying (Modulación en Cuadratura por Salto de Fase)
ROM	Read-Only Memory (Memoria de Sólo Lectura)
SS	Subscriber Station (Estación Subscriptora)
SISO	Single Input Single Output (Única Entrada Única Salida)
UL	Uplink (Enalce Ascendente)
UMTS	Universal Mobile Telecommunications System (Sistema Universal de Telecomunicaciones Móviles)
VHDL	Very High Speed Integrated Circuit Hardware Description Language (Lenguaje de Descripción de Soportes Físicos para Circuitos Integrados Muy Rápidos)
WiMAX	Worldwide Interoperability for Microwave Acess (Interoperabilidad Mundial para Acceso por Microondas)
xDSL	Digital Subscriber Line (Línea de Subscripción Digital)

Capítulo 1

Introducción

1. Antecedentes del proyecto

A lo largo de los años, los sistemas de telecomunicación han ido evolucionando como consecuencia de la aparición de nuevas necesidades por parte de la sociedad. Cada vez son más demandados sistemas en los que, por una parte se ofrezcan servicios de banda ancha, y por otra, se permita que el usuario goce de movilidad. Bajo estas dos demandas, banda ancha y movilidad, el grupo de trabajo IEEE (*Institute of Electrical and Electronics Engineers*) 802.16 ha desarrollado una serie de estándares donde se recopilan todas las tecnologías para el acceso inalámbrico de banda ancha a nivel metropolitano. Así, esta tecnología, será una dura competencia tanto para los sistemas cableados (DSL (*Digital Subscriber Line*), fibra óptica…), como para las comunicaciones móviles basadas en la tecnología 3G (UMTS (*Universal Mobile Telecommunications System*)).

Este tipo de tecnología está basada en el empleo de una modulación multiportadora llamada "*Ortogonal Frequency Division Multiplexing*" (OFDM). Esta modulación presenta una gran robustez ante canales selectivos en frecuencia, ya que divide el espectro de frecuencias en canales que presentan un desvanecimiento prácticamente plano, posibilitando así una sencilla igualación de los mismos. Además, gracias a la existencia de eficientes algoritmos para el cálculo de la Transformada Rápida de Fourier (FFT (*Fast Fourier Transform*)), esta modulación es posible implementarla de forma sencilla en la práctica.

Por otra parte, se busca tener sistemas inalámbricos de gran capacidad. Para ello, en ciertos estándares desarrollados por el grupo de trabajo IEEE 802.16 se permite añadir, tanto en recepción como en transmisión, más de una antena, de manera que es posible la implementación de sistemas multiantena, "*Multiple Input – Multiple Output*" (MIMO). Con este tipo de sistemas, se emplean técnicas de diversidad espacial, que tras un apropiado procesado de señal, permiten aumentar la capacidad del sistema, además de solucionar los posibles desvanecimientos de la señal en el canal, como consecuencia de los

posibles caminos que las réplicas de ésta pueden tomar entre el emisor y el receptor.

De la unión de las características que ofrece la modulación OFDM y las técnicas multiantena MIMO, se pueden implementar sistemas muy robustos frente a desvanecimientos selectivos en frecuencia y con una alta tasa de transmisión. Este tipo de sistemas, conocidos como sistemas MIMO-ODFM, son muy sensibles a la sincronización en tiempo y en frecuencia. El hecho de no saber en qué instante de tiempo comienza una trama o si la trama enviada sufre un desplazamiento en frecuencia puede suponer una importante degradación en la calidad del sistema si no se corrige. Por ello, el presente proyecto pretende completar el diseño de un sistema MIMO-OFDM basado en el estándar IEEE 802.16d-2004 (*Wimax fijo*) en el que se han obviado estos dos efectos perjudiciales para la calidad del sistema. También, en el sistema MIMO-OFDM modelado se obvió la existencia de canal por lo que en el presente proyecto se simulará tal canal para posteriormente corregir los efectos que éste ocasiona sobre la señal emitida.

2. Objetivos

Este proyecto vino motivado por la necesidad de completar el proyecto realizado por los autores David Díaz Martín y Roberto Prieto Alonso. En su proyecto trataron de diseñar en hardware, simular y validar un sistema de comunicación completo (emisor y receptor) para el estándar IEEE 802-16d-2004 utilizando MIMO 2x2, OFDM en la capa física y BPSK (*Binary Phase Shift Keying*) para la modulación de datos. Trataron de describir en hardware este sistema de comunicación utilizando la plataforma de desarrollo para FPGA's (*Field Programmable Gate Arrays*) basada en Simulink®: VHS-ADC Virtex-4 de Lyrtech® junto a *Xilinx System Generator for DSP*. El diseño que realizaron suponía la no existencia de canal entre el emisor y el receptor del sistema, además de una perfecta sincronización de los mismos. Estas circunstancias no se producen en la práctica, por lo que se pretende diseñar los módulos relativos a la estimación de canal, a la sincronización en frecuencia y a la sincronización temporal. El diseño de estos módulos completará el trabajo iniciado por los dos autores anteriormente citados.

3. Descripción de contenidos

El presente proyecto, como ya comentamos, tiene como objetivo el diseño de los módulos de estimación de canal, sincronización en frecuencia y sincronización temporal, carentes en el receptor del sistema diseñado por los autores anteriormente citados. Para poder comprender como estos módulos están desarrollados es preciso tener una base teórica de cómo los sistemas MIMO-OFDM funcionan y conocer el estándar IEEE 802.16d-2004 (*Wimax fijo*). Por ello, en el capítulo 2 se describen los fundamentos teóricos de cómo funciona un sistema MIMO-OFDM y en el capítulo 3 se describen las características de la capa física del estándar de Wimax que tienen mayor relevancia a la hora de estimar el canal.

Una vez adquiridos los fundamentos teóricos de cómo funciona un sistema de comunicaciones de este tipo, en el capítulo 4 se hará una descripción de la herramienta utilizada para el diseño de los módulos a añadir al receptor del sistema.

En los capítulos 5 y 6, se describirán los algoritmos empleados en el modelado hardware de los módulos a diseñar. Cada uno de estos capítulos, estará estructurado de tal manera que, en la primera parte del capítulo se explique de manera teórica el algoritmo a desarrollar, en una segunda parte se explique el hardware descrito, y en una tercera y última parte, se validen los diseños y se comenten los resultados obtenidos.

En el capítulo 7, se hará una descripción de las principales conclusiones obtenidas tras la realización de este proyecto, orientadas a determinar que trabajos futuros son necesarios para la mejora del sistema de comunicaciones modelado.

Por último, en el capítulo 8 se detalla el coste de este proyecto mediante la elaboración de un presupuesto.

Capítulo 2

Fundamentos teóricos de OFDM y MIMO

1. OFDM (Ortogonal Frequency Division Multiplexing)

Este tipo de modulación multiportadora está basado en la división del espectro de frecuencias en una serie de subcanales (a cada subcanal se le atribuye una subportadora), cada uno de los cuales presenta un desvanecimiento aproximadamente plano [34]. Así, el conjunto de subcanales que dividen el espectro, constituyen lo que se conoce como símbolo OFDM. El hecho de que cada uno de estos subcanales presente un desvanecimiento plano, supone que el proceso de igualación en el receptor se pueda realizar de forma sencilla. Cada uno de estos subcanales se solapa y se ortogonaliza para obtener una alta eficiencia espectral. Para evitar la pérdida de ortogonalidad entre las subportadoras ante un canal dispersivo, lo que se hace es añadir un prefijo cíclico que, como veremos, únicamente supondrá una copia de la parte final del símbolo OFDM a transmitir. La introducción de este prefijo cíclico supondrá una perdida en la relación señal a ruido pero será necesaria para mantener la ortogonalidad, y evitar así la interferencia entre símbolos y entre subportadoras.

A continuación explicaremos el modelo continuo de la modulación OFDM, para posteriormente deducir el modelo discreto de la misma, ya que es la versión que nos interesa para el tipo de sistema de comunicación que consideraremos en el presente proyecto.

1.1 OFDM continuo

En el modelo OFDM continuo, la señal en banda base que se obtiene es de la forma [101]:

$$s(t) = \sum_{k=0}^{N-1} \sum_{l=-\infty}^{\infty} S^l(k) \emptyset_k(t - lT) \qquad (2.1)$$

donde $S^l(k)$ se corresponde con un símbolo complejo de la constelación que se emplea para la modulación de datos(QAM, QPSK, BPSK, etc.). Así, este símbolo es transportado por la subportadora k-ésima del l-ésimo símbolo OFDM. T es el tiempo que dura el símbolo OFDM y N el número de subportadoras. Cada símbolo complejo de la constelación, esta multiplicado por una función base $\emptyset_k(t - lT)$. El conjunto de funciones base que multiplican a los distintos símbolos de la constelación, forman una base ortogonal constituida por un conjunto de N exponenciales complejas acotadas temporalmente por una ventana de T segundos de duración. Esta ventana de acotación, normalmente suele ser un pulso rectangular, por lo que cada función base no es más que un pulso rectangular modulado por una subportadora de frecuencia f_k. Así, la señal ODFM en banda base se puede expresar de la forma:

$$s(t) = \frac{1}{\sqrt{T}} \sum_{k=0}^{N-1} \sum_{l=-\infty}^{\infty} S^l(k) e^{-j2\pi f_k (t-lT)} \qquad (2.2)$$

donde $\frac{1}{\sqrt{T}}$ es un factor usado para la normalizar la energía de las funciones base a la unidad.

Para caracterizar a las subportadoras de un símbolo OFDM, es preciso saber qué valores puede tomar f_k. Para ello, se ha de calcular previamente la separación entre subportadoras. En la realización de los cálculos, supondremos un canal con un ancho de banda disponible de W Hz. Así, como el canal de anchura W Hz se divide entre el número de subportadoras que constituyen el símbolo OFMD, N, la separación entre portadoras es $\Delta f = W/N$ Hz. Si queremos que exista ortogonalidad entre subportadoras, la duración del símbolo OFDM debe de ser $T = 1/\Delta f = N/W$ segundos. Finalmente, el conjunto de valores que pueden tomar las distintas subportadoras de un símbolo OFDM se calcula del modo:

$$f_k = f(k) = \frac{1}{T} = \frac{kW}{N}, \quad k = 0, \dots, N-1 \qquad (2.3)$$

Sustituyendo (2.3) en (2.2) la señal OFDM en banda base queda de la forma.

$$s(t) = \frac{1}{\sqrt{T}} \sum_{k=0}^{N-1} \sum_{l=-\infty}^{\infty} S^l(k) e^{-j2\pi \frac{kW}{N}(t-lT)} \qquad (2.4)$$

Como podemos apreciar en la figura (2.1), la respuesta en frecuencia de cada subportadora es nula para los máximos de la respuesta en frecuencia del resto de subportadoras. De esta manera observamos de una manera gráfica, cómo la ortogonalidad de las subportadoras evita la interferencia entre símbolos.

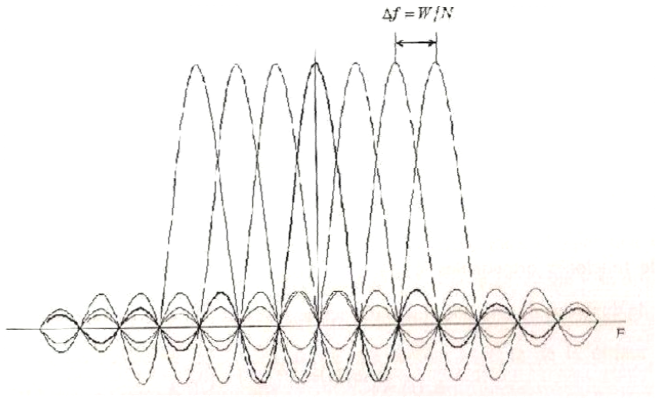


Figura 2.1: Espectro de las subportadoras de un símbolo OFDM [34]

Para tener una visión gráfica y comprender conceptualmente como funciona esta modulación se muestra en la figura (2.2) el diagrama Tiempo-frecuencia de una serie de símbolos OFDM. Como podemos ver, con esta modulación es posible enviar en un mismo intervalo de tiempo (duración del símbolo OFDM) información en todas las subportadoras sin que exista interferencia, ya que éstas son ortogonales.

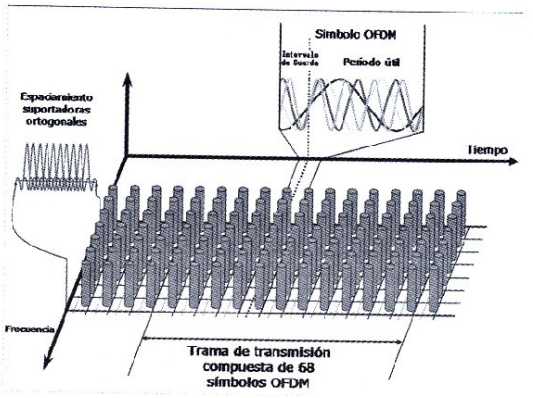


Figura 2.2: Diagrama tiempo-frecuencia de una serie de símbolos OFDM [34]

1.2 OFDM discreto

Para obtener la versión discreta de esta modulación, no hay más que aplicar el teorema de muestreo de Nyquist a la versión continua anteriormente explicada. Por un lado, la duración del símbolo OFDM será la misma que para el caso continuo, es decir, $T = 1/\Delta f = N/W$ segundos. Esto es equivalente a decir que la separación entre portadoras Δf es la misma que para el caso continuo, de manera que se mantiene la ortogonalidad entre las mismas. Por otro lado, si se aplica el teorema de muestreo, la frecuencia de muestreo mínima deberá ser igual al ancho de banda disponible, es decir, $f_s = W$ o lo que es lo mismo, que el periodo de muestreo es $T_s = 1/W$. De la unión de la expresión para el tiempo de símbolo y el periodo de muestreo, se puede deducir la expresión del tiempo de símbolo OFDM en función de periodo de muestreo de la siguiente manera: $T = NT_s = N(1/W)$. Así, podemos ver que para un símbolo OFDM se toman N muestras, o en otras palabras, a cada subportadora le corresponde una muestra cuando no existe sobremuestreo.

Muestreando las funciones base del caso continuo obtendremos su equivalente discreto. Para ello, muestreamos a la frecuencia de muestreo indicada anteriormente. El proceso a seguir es el siguiente:

$$e^{j2\pi\frac{kW}{N}(t-lT)} \rightarrow t = nT_s,\ \ T = N/W \rightarrow e^{j2\pi\frac{kW}{N}\left(nT_s - l\frac{N}{w}\right)} \rightarrow T_s = 1/W \rightarrow$$

$$\rightarrow e^{j2\pi\frac{k}{N}(n-lN)}, \qquad n = 0, \dots, N-1 \qquad (2.5)$$

Ahora, utilizando la base de funciones ortogonales discreta deducida en (2.5), obtenemos el equivalente discreto para el l-ésimo símbolo OFDM mediante la siguiente expresión:

$$s^l[n] = \sum_{k=0}^{N-1} S^l(k)\phi_k[n - lN] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} S^l(k) e^{j2\pi \frac{k}{N}(n-lN)} \qquad (2.6)$$

Si particularizamos para un símbolo en concreto se tiene:

$$s[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} S(k) e^{j2\pi \frac{k}{N} n} \qquad (2.7)$$

El resultado obtenido en la expresión (2.7) es de gran importancia, ya que la obtención del símbolo OFDM en banda base, es simplemente realizar una IDFT (*Inverse Discrete Fourier Transform)* de los símbolos de la constelación a transmitir. Por otro lado, en demodulación habrá que realizar la operación inversa, es decir, se realizará una operación de DFT (*Discrete Fourier Transform*) para obtener los símbolos de la constelación transmitida. El hecho de que la modulación y demodulación se lleve a cabo mediante las operaciones de IDFT y DFT supone una gran ventaja. Esto es debido a la existencia de algoritmos como la FFT (*Fast Fourier Transform*) que permiten realizar las operaciones de IDFT y DFT de manera rápida y eficiente. Para aplicar estos algoritmos, se requerirá que el número de subportadoras sea potencia de dos, cosa común en la mayoría de los casos.

1.3 Prefijo cíclico

La ortogonalidad de las subportadoras de un símbolo OFDM en transmisión, se pierde en recepción debido a las características que posee el canal radio. Un canal dispersivo y el efecto multicamino, afectan a la ortogonalidad de las subportadoras del símbolo OFDM, de manera que aparecen los efectos de interferencia entre símbolos e interferencia entre subportadoras.

Para evitar la aparición de interferencia entre símbolos, se puede introducir un intervalo de guarda al comienzo de cada símbolo OFDM. Este intervalo de guarda, deberá tener una duración mayor a la respuesta impulsional del canal para llevar a cabo su función. De esta manera, a la duración del símbolo OFDM habrá que sumarle la duración de intervalo de guarda.

En el intervalo de guarda tendrá lugar la interferencia entre símbolos, que será posteriormente eliminada en recepción. Sin embargo, la inserción del intervalo de guarda no supone la eliminación de la interferencia entre subportadoras. Esto se debe a que el efecto multicamino del canal, puede producir que no haya una diferencia de periodos entero entre las subportadoras de las réplicas recibidas en la parte útil del símbolo OFDM transmitido, provocando la pérdida de ortogonalidad entre las subportadoras. Este efecto perjudicial se soluciona mediante la inclusión, en el intervalo de guarda, de un prefijo cíclico, que no es más que la copia de la parte final del símbolo OFDM en tal espacio temporal. Con este prefijo cíclico, las subportadoras que han sufrido cierto retardo por el efecto multicamino, poseerán un número entero de periodos en la parte útil del símbolo OFDM. Así, la ortogonalidad entre las subportadoras, se mantendrá siempre y cuando la longitud del prefijo cíclico sea mayor que la respuesta impulsional del canal.

En la figura (2.3), se ilustra cómo se añade el prefijo cíclico, de manera que podemos observar cual es la parte útil del símbolo OFDM (la parte de datos) y cuál es la parte que corresponde al prefijo cíclico, el cual será eliminado en recepción. Vemos como las muestras finales del símbolo se copian al comienzo del mismo.

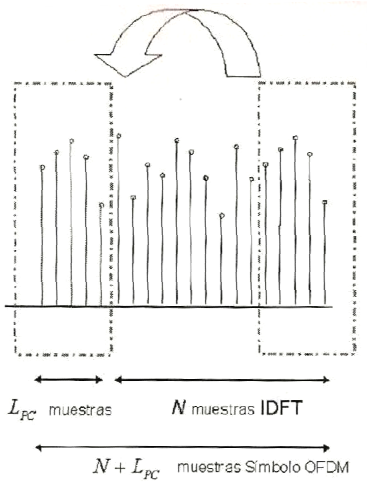


Figura 2.3: Formación del prefijo cíclico de un símbolo OFDM [34]

1.4 Sistema OFDM

Una vez explicada la modulación OFDM, vamos a describir el diagrama de bloques simplificado de un sistema de comunicaciones que emplea esta modulación. En la figura (2.4), se pueden observar los bloques que van a modular (emisor) y demodular (receptor) la información de la fuente transmisora.

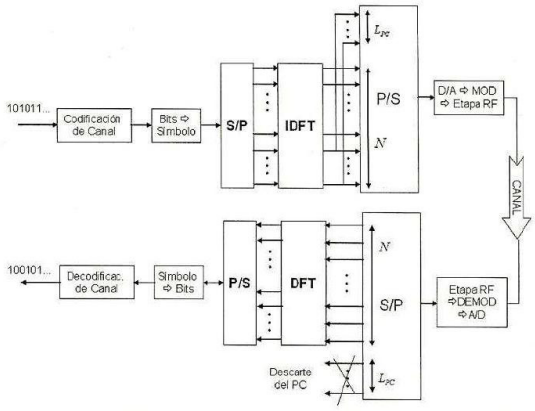


Figura 2.4: Diagrama de bloques básico de un sistema OFDM [34]

En la parte del emisor, se codifica la información de la fuente mediante un codificador de canal. A continuación, se agrupan los bits para poder asignarlos a un símbolo de la constelación a transmitir. Los símbolos de la constelación a transmitir se distribuyen en paralelo para que se realice sobre ellos la operación de IDFT de manera que se obtiene el símbolo OFDM en banda base. Al símbolo OFDM formado, se le añade el prefijo cíclico comentado en la sección anterior (de L_{pc} muestras de la figura), y se serializa para posteriormente ser enviado al canal radio modulado por una portadora de radio frecuencia. En la parte del receptor, se realizarán todas las operaciones comentadas pero en sentido inverso.

La cadena de transmisión mostrada en la figura (2.4) trata de un sistema SISO (*Single Input Single Output*). Sin embargo, si queremos considerar un sistema MIMO (*Multiple Input Multiple Output*), se requerirá repetir este esquema tantas veces como antenas se consideren. Notar que en el presente proyecto, trataremos un sistema en el cual tenemos dos antenas en recepción y dos antenas en transmisión (sistema 2x2 MIMO), por lo que tendremos este esquema por duplicado.

2. MIMO (Multiple Input Multiple Output)

Hoy en día, se demandan cada vez más sistemas inalámbricos de gran calidad de servicio y velocidad, por lo que aparece la necesidad de nuevas técnicas que mejoren la eficiencia espectral del sistema de comunicaciones y las prestaciones de los radioenlaces. Para ello, apareció una nueva técnica de diversidad espacial conocida como MIMO, la cual emplea varias antenas, tanto en un extremo del radioenlace como en el otro, es decir, tanto en el emisor como en el receptor. Este tipo de técnica es la empleada en el presente proyecto, por lo que, en esta sección, nos dispondremos a describir sus fundamentos básicos.

2.1 Diversidad espacial

Las técnicas de diversidad, tratan de resolver los efectos perjudiciales que se producen sobre la señal transmitida en un radioenlace, como consecuencia de los desvanecimientos del canal y el efecto multicamino. Así, la señal transmitida viajará a través de medios o canales independientes hasta el receptor.

Existen tres principales técnicas de diversidad: diversidad temporal, diversidad en frecuencia y diversidad espacial. Estos tres tipos de diversidad se pueden combinar para dar lugar a técnicas híbridas de diversidad. Sin embargo, en la presente sección solo nos centraremos en la diversidad espacial.

La técnica de diversidad espacial se aplica en los sistemas multiantena, y se basa en la emisión y/o recepción de señales mediante una serie de antenas separadas una distancia mayor que la distancia de coherencia del canal. Esta distancia de coherencia, es la distancia mínima que tiene que haber entre las distintas antenas del sistema para que los desvanecimientos producidos en los distintos canales del sistema sean independientes.

En la figura (2.5), podemos apreciar la composición de un sistema que aplica diversidad espacial. Concretamente, se trata de un sistema MIMO, el cual está formado por M_T antenas transmisoras y M_R antenas receptoras, entre las cuales se establecen una serie de canales (canal MIMO).

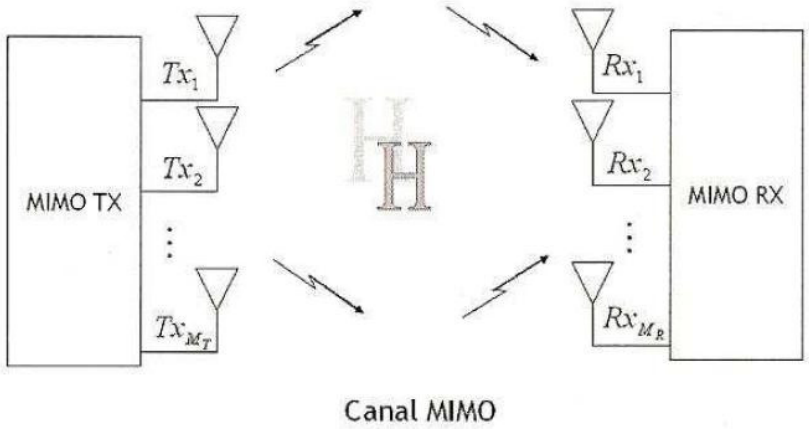


Figura 2.5: Diagrama de bloques básico de un sistema MIMO [34]

A raíz de este esquema, podemos distinguir distintas técnicas de diversidad espacial, tanto en el emisor como en el receptor. Estas técnicas de diversidad, no son de gran relevancia en el cometido de este proyecto, por lo que solo nos dispondremos simplemente a comentarlas brevemente.

La técnicas de diversidad espacial en recepción, simplemente hacen referencia a como se debe procesar la información recibida en las distintas antenas. Entre este tipo de técnicas de diversidad destacan: diversidad por selección (se selecciona la antena con mayor SNR recibida), diversidad por conmutación (se conmuta a otra antena cuando la señal recibida sobrepasa un umbral) y diversidad por combinación (la señal usada en detección es la combinación lineal de las distintas réplicas, cada una de las cuales tiene un peso). En la figura (2.6) podemos apreciar esquemáticamente las técnicas enumeradas.

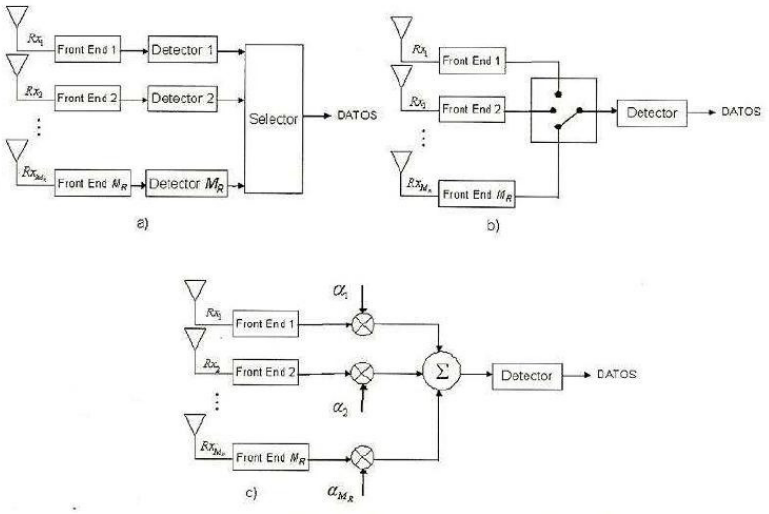


Figura 2.6: Técnicas de diversidad espacial en recepción [34]

Las técnicas de diversidad en transmisión, hacen referencia al procesado previo que debe sufrir la señal antes de ser transmitida por las distintas antenas del transmisor. Entre los transmisores que emplean estas técnicas destacamos: Trasmisores basados en esquemas realimentados, transmisores basados en esquemas que emplean información de entrenamiento (es el caso del presente proyecto que utilizará esa información para la estimación de canal entre emisor y receptor) y transmisores basados en esquemas combinados (se combina la diversidad en transmisión con codificación de canal). Por último, comentar que el uso del codificador de canal en el último tipo de transmisor enumerado, proporciona diversidad a costa de perder ancho de banda. Para solucionar el problema, existen esquemas de transmisores que emplean una codificación, conocida como codificación espacio-tiempo, que permite obtener la diversidad deseada sin perder ancho de banda disponible.

2.2 Multiplexado espacial

Con el empleo de técnicas MIMO, además de obtener ganancia por diversidad espacial, es posible aumentar la capacidad del sistema de comunicación. Para ello, nos basamos en el concepto de multiplexado espacial. Esta técnica parte de la premisa de que con el uso de múltiples antenas en transmisión y recepción (MIMO), es posible establecer múltiples canales de datos en una misma banda de frecuencia, de manera que se aumenta la capacidad del

sistema. Gracias a esta técnica, se consigue aumentar la capacidad sin necesidad de un aumento de ancho de banda ni de un aumento de potencia.

El principio de funcionamiento de esta técnica se ilustra en la figura (2.7). En ella, podemos observar como los símbolos a transmitir se disponen en paralelo para formar subtramas de símbolos, las cuales se transmiten por cada antena en la misma banda de frecuencias. Para la correcta aplicación de esta técnica, hay que tener en cuenta que será preciso que el número de antenas receptoras siempre sea mayor o igual al número de antenas transmisoras.

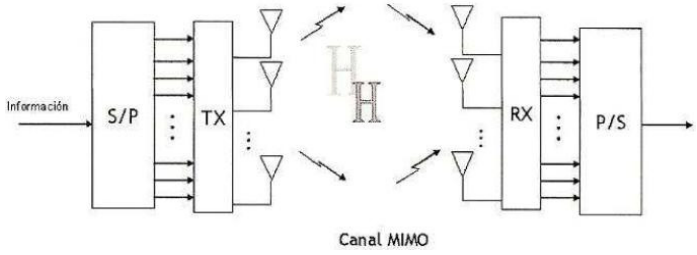


Figura 2.7: Multiplexado espacial [34]

Existen estudios que demuestran que la capacidad de un sistema que usa técnicas MIMO es mayor que la de los sistemas convencionales con una sola antena en transmisión y recepción (SISO). Un ejemplo de ello, es el artículo [37], el cual habla de la capacidad de los canales MIMO. De la lectura de este artículo se extrae que la capacidad de un canal MIMO aumenta linealmente con el número de antenas usadas en el sistema.

3. Sistema MIMO-OFDM

En la siguiente sección, trataremos de unificar los conceptos desarrollados en secciones anteriores (MIMO y OFDM). Esta unificación de conceptos, dará lugar a la descripción del tipo de sistema que se empleará en el presente proyecto, un sistema MIMO-OFDM. En un primer lugar, partiremos de un sistema SISO-OFDM para extenderlo a la versión MIMO al final de la sección.

3.1 Sistema SISO-OFDM

En este tipo de sistemas, tan solo tendremos una antena en transmisión y una antena en recepción. El transmisor enviará a través de su única antena el símbolo OFDM deducido en (2.6). Así, la información de la fuente se agrupará en grupos de N símbolos complejos, cada uno de los cuales pertenecerá a la constelación de un tipo de modulación de datos determinado (BPSK, QPSK, etc.).A su vez, cada símbolo complejo será transportado por una de las N subportadoras que constituyen el símbolo OFDM. Por tanto, la señal temporal tras aplicar la modulación OFDM es de la forma:

$$s^l[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} S^l(k) e^{j2\pi \frac{k}{N}(n-lN)} \qquad n,k = 0,\dots,N-1 \qquad (2.8)$$

donde $S^l(k)$ se corresponde con un símbolo complejo de la constelación que se emplea para la modulación de datos (QAM, QPSK, BPSK, etc.), el cual es transportado en la k-ésima subportadora. $s^l[n]$ hace referencia a las muestras temporales del l-ésimo símbolo OFDM, siendo N el número de muestras del símbolo cuando no existe sobremuestreo. Como ya vimos, la formación del símbolo OFDM en banda base, únicamente implicaba la realización de una IDFT (*Inverse Discrete Fourier Transform*) sobre el conjunto de N símbolos complejos que se desea transmitir.

Así, la expresión (2.8), también se puede expresar de forma matricial de la siguiente manera:

$$\underline{s^l} = \underline{\underline{F^H}}\,\underline{S^l} \qquad (2.9)$$

donde $\underline{s^l}$ es un vector que contiene las muestras temporales del símbolo OFDM banda base ($\underline{s^l} = [s^l(0),\dots,s^l(N-1)]^T$), $\underline{S^l}$ es un vector que contiene los N símbolos complejos que se desean transmitir ($\underline{S^l} = [S^l(0),\dots,S^l(N-1)]^T$) y $\underline{\underline{F}}$ es una matriz de dimensión NxN que realiza la operación de DFT sobre los símbolos complejos. Concretamente, la matriz $\underline{\underline{F}}$ es de la forma:

$$\underline{\underline{F}} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-j\frac{2\pi}{N}1\cdot 1} & \dots & e^{-j\frac{2\pi}{N}(N-1)\cdot 1} \\ \dots & \dots & \dots & \dots \\ 1 & e^{-j\frac{2\pi}{N}1\cdot(N-1)} & \dots & e^{-j\frac{2\pi}{N}(N-1)(N-1)} \end{bmatrix} \qquad (2.10)$$

Notar que, en la expresión (2.9), la matriz $\underline{\underline{F^H}}$ se corresponde con la matriz compleja conjugada de la matriz $\underline{\underline{F}}$, ya que la operación que se realiza es una IDFT.

Una vez determinada la señal que se transmite por el canal, vamos a ver qué señal se recibe en el otro extremo del radioenlace. Teniendo en cuenta que la modulación OFDM divide el canal en una serie de subcanales con desvanecimiento aproximadamente plano, la señal recibida en el dominio de la frecuencia, puede expresarse para cada subcanal "k" de la forma:

$$R^l(k) = H(k)S^l(k) + W^l(k) \qquad k = 0, \dots, N-1 \qquad (2.11)$$

donde $R^l(k)$ se corresponde con el símbolo complejo recibido del l-ésimo símbolo OFDM transportado en la k-ésima subportadora, $H(k)$ es el coeficiente de canal en la k-ésima subportadora y $W^l(k)$ es la contribución de ruido en la k-ésima subportadora.

Si consideramos la contribución de las N subportadoras de un símbolo OFDM, la ecuación (2.11) supone un sistema de N ecuaciones, las cuales podemos expresar de forma matricial de la siguiente manera:

$$\underline{R^l} = \underline{\underline{H}}\,\underline{S^l} + \underline{W^l} \qquad (2.12)$$

donde $\underline{R^l}$ es un vector donde se recogen los N símbolos complejos recibidos, los cuales son transportados por el l-ésimo símbolo OFDM ($\underline{R^l} = [R^l(0), \dots, R^l(N-1)]^T$), $\underline{S^l}$ es un vector donde se recogen los N símbolos complejos transmitidos en el l-ésimo símbolo OFDM ($\underline{S^l} = [S^l(0), \dots, S^l(N-1)]^T$) y $\underline{\underline{H}}$ es una matriz diagonal donde los elementos de la diagonal representan los coeficientes de canal en cada subportadora "k". Concretamente, $\underline{\underline{H}}$ es de la forma:

$$\underline{\underline{H}} = \begin{bmatrix} H(0) & 0 & 0 & 0 \\ 0 & H(1) & 0 & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & H(N-1) \end{bmatrix} \qquad (2.13)$$

Para calcular los coeficientes de canal representados en la diagonal de $\underline{\underline{H}}$, habrá que calcular la DFT de N putos de los coeficientes de canal en el dominio del tiempo, es decir, de $h[n]$. Si se toma que $h[n]$ tiene una longitud mínima de L muestras (tamaño que, como ya vimos, deberá ser menor que el número de

muestras del prefijo cíclico del símbolo OFDM), las cuales son menores que el número de puntos de la DFT (N), habrá que ampliar el número de muestras de $h[n]$ hasta N muestras mediante la inserción de muestras nulas, y así poder hacer el cálculo de la DFT.

3.2 Sistema MIMO-OFDM

Para caracterizar el sistema MIMO-OFDM [34], simplemente habrá que realizar una extensión del caso SISO anteriormente descrito. Para describir el sistema MIMO de forma genérica, definimos M_T antenas en transmisión y M_R antenas en recepción, de manera que el canal MIMO está formado por $M_T \cdot M_R$ canales SISO. Para poder definir los canales SISO que componen el canal MIMO, definimos una serie de índices que nos indican a qué antenas nos referimos. Así, el canal SISO entre la j-ésima antena transmisora (j= $0, \dots, N-1$) y la i-ésima antena receptora (i= $0, \dots, N-1$) se define como $h_{i,j}[n]$. Como vimos en otros apartados, estos canales deberán tener una longitud máxima L (etapas), no superior a la longitud del prefijo cíclico.

Por cada antena transmisora se transmitirá el un símbolo OFDM, el cual estará constituido por N muestras temporales, tal y como se demostró en el apartado de OFDM. Las muestras temporales del símbolo OFDM, se obtienen de hacer la IDFT a un conjunto N símbolos complejos. Así, la señal transmitida por la j-ésima antena transmisora se puede expresar matricialmente de la forma:

$$\underline{s^l}_j = \underline{\underline{F^H}}\,\underline{S^l}_j \qquad (2.14)$$

donde $\underline{s^l}_j$ es un vector donde se recogen la N muestras temporales que constituyen el l-ésimo símbolo OFDM transmitido por la j-ésima antena transmisora ($\underline{s^l}_j = \left[s_j^l(0), \dots, s_j^l(N-1)\right]^T$) y $\underline{S^l}_j$ es un vector donde se recogen los N símbolos complejos transmitidos sobre el l-ésimo símbolo OFDM en la j-ésima antena transmisora ($\underline{S^l}_j = \left[S_j^l(0), \dots, S_j^l(N-1)\right]^T$).

Para obtener la señal recibida en una antena receptora "i", simplemente hemos de considerar los símbolos OFDM transmitidos en cada una de las M_T antenas transmisoras. Por tanto, la ecuación (2.11) de la versión SISO, se extiende de la siguiente manera para dar lugar a la señal recibida en la i-ésima antena receptora en el dominio frecuencial:

$$R_i^l(k) = \sum_{j=1}^{M_T} H_{i,j}(k) S_j^l(k) + W_i^l(k) \qquad i = 1, \dots, M_R \qquad (2.15)$$

donde $S_j^l(k)$ se corresponde con un símbolo complejo enviado por la j-ésima antena transmisora sobre la k-ésima subportadora del l-ésimo símbolo OFDM, $W_i^l(k)$ se corresponde con muestras de ruido gaussiano y $H_{i,j}(k)$ es la respuesta en frecuencia del canal entre la antena transmisora "j" y la receptora "i" en la portadora "k". Como se puede deducir, $R_i^l(k)$ es el resultado de la suma de los símbolos complejos transmitidos en cada antena transmisora una vez han atravesado el canal MIMO.

La respuesta en frecuencia de los canales SISO de L etapas que componen el canal MIMO, se calculará tal y como se explicó en el apartado anterior, es decir, mediante la aplicación de una DFT de N puntos a la respuesta impulsional de cada canal SISO.

Para obtener expresión de la señal recibida para el sistema MIMO completo, únicamente deberemos considerar todas las antenas receptoras. Así, si consideramos todas las antenas que intervienen en el sistema MIMO, podemos extender la ecuación (2.15) para dar lugar a la ecuación que nos proporciona la respuesta en frecuencia de la señal recibida en la k-ésima subportadora:

$$\underline{R^l}(k) = \underline{\underline{H}}(k)\underline{S^l}(k) + \underline{W^l}(k) \qquad (2.16)$$

donde $\underline{R^l}(k)$ es un vector donde se recogen los símbolos complejos recibidos en cada antena receptora, transportados en la k-ésima subportadora del l-ésimo símbolo OFDM ($\underline{R^l}(k) = \left[R_1^l(k), \dots, S_{M_R}^l(k)\right]^T$), $\underline{W^l}(k)$ es un vector que recoge las muestras de ruido gaussiano en las M_R antenas receptoras para la k-ésima subportadora y $\underline{\underline{H}}(k)$ es una matriz de dimensión $M_R \, x \, M_T$ que contiene, en el dominio frecuencial, los coeficientes de canal de los $M_R M_T$ canales SISO ($H_{i,j}(k)$) que componen el canal MIMO en la k-ésima subportadora.

Para describir el sistema MIMO-OFDM, hay que considerar todos los subcanales en los que la modulación OFDM ha dividido el ancho de banda disponible. Así, la expresión descrita en (2.16) deriva en la expresión (2.17), la cual considera todos los subcanales del espectro y todas las antenas receptoras. Así, en la expresión (2.17) se recoge la relación entrada-salida para el sistema MIMO-OFDM completo:

$$\underline{R^l}_{TOTAL} = \underline{\underline{H}}_{TOTAL}\underline{S^l}_{TOTAL} + \underline{W^l}_{TOTAL} \qquad (2.17)$$

donde $\underline{R^l}_{TOTAL}$ es un vector de símbolos complejos recibidos ($\underline{R^l}_{TOTAL} = \left[\underline{R^l}(0), \dots, \underline{R^l}(N-1)\right]^T$), $\underline{S^l}_{TOTAL}$ es un vector de símbolos complejos transmitidos ($\underline{S^l}_{TOTAL} = \left[\underline{S^l}(0), \dots, \underline{S^l}(N-1)\right]^T$), y $\underline{W^l}_{TOTAL}$ es un vector que representa muestras de ruido gaussiano. $\underline{\underline{H}}_{TOTAL}$ es una matriz diagonal que aglutina todos los coeficientes de canal, en frecuencia, del canal MIMO. Los elementos de su diagonal hacen referencia a la matriz $\underline{\underline{H}}(k)$ de la expresión (2.16), de manera que para el primer elemento de la diagonal se evalúa $\underline{\underline{H}}(k)$ en $k = 0$, para el segundo se evalúa $\underline{\underline{H}}(k)$ en $k = 1$, y así sucesivamente hasta cubrir todas las subportadoras cuando se llega a $k = N-1$ ($\underline{\underline{H}}_{TOTAL} = diag\left(\underline{\underline{H}}(0),\ \underline{\underline{H}}(1), \dots, \underline{\underline{H}}(N-1)\right)$).

Capítulo 3

El estándar IEEE 802.16 (Wimax)

1. Introducción

En el siguiente capítulo, trataremos de describir los fundamentos básicos del estándar en el cual se basa el sistema de comunicación del presente proyecto. Concretamente, describiremos el estándar IEEE 802.16d-2004 (*WiMAX fijo*) [38], el cual pertenece a la familia de estándares de acceso inalámbrico IEEE 802.16 conocidos como WiMAX (*Worldwide Interoperability for Microwave Access*).

WiMAX es un estándar en el cual se definen las tecnologías de acceso inalámbrico de banda ancha "*de última milla*". Esta tecnología ofrece prestaciones similares a los sistemas cableados de banda ancha (como por ejemplo, la fibra óptica o los sistemas xDSL), por lo que implica una alternativa interesante para el acceso a la banda ancha.

Una característica importante de esta tecnología, es la facilidad con la que se puede desplegar. Así, los costes relativos su implantación son considerablemente inferiores a los de los sistemas cableados. Simplemente, con las ondas de radio emitidas por una estación base (*Base Stations*, BS) se podrá dar servicio de banda ancha a múltiples estaciones subscriptoras (*Subscriber Stations*, SS) en un amplia área o celda (48 km de radio), como podemos observar en la ilustración de la figura (3.1).

Figura 3.1: Esquema simplificado de la tecnología Wimax [34]

El hito que impulsó que la industria comenzara a creer en esta tecnología fue la aparición del estándar de WiMAX móvil (IEEE 802.16e-2005). Para los operadores y fabricantes de equipos, la irrupción de este nuevo estándar, implicaba mayores beneficios económicos. Esto posibilitó la inversión en este tipo de tecnología para su desarrollo [34]. Sin embargo, el presente proyecto considera la versión de WiMAX fijo (IEEE 802.16d-2004), ya que trata de completar las líneas de trabajo marcadas por los autores del sistema de comunicación a completar.

2. Wimax fijo (IEEE 802.16.d-2004)

Este estándar trata de proporcionar acceso fijo inalámbrico de banda ancha a una tasa de transmisión de datos de hasta 75 Mbps con un alcance entre 5 y 10 km cuando no hay visión directa (obstáculos entre emisor y receptor). En caso de visión directa, esta tasa de transmisión se mantiene en un radio de 50 km [38]. Estas tasas de transmisión son posibles a través del canal inalámbrico, ya que esta tecnología está basada en la modulación OFDM. Debido a la robustez de esta modulación en entornos sin visión directa (*Non Line of Sight*, NLOS), es posible llegar a estas elevadas tasas de transmisión para un amplio número de usuarios.

El estándar marca la banda de frecuencia de funcionamiento entre 2 y 11 GHz y proporciona servicios de banda ancha a múltiples usuarios con una escalabilidad entre canales de 1.5 MHz a 20 MHz [38]. Esto posibilita que se

puedan proporcionar distintas calidades de servicio (*Quality of Service*, QoS) a niveles superiores a la capa física de la arquitectura de red.

En el presente proyecto, para el cumplimiento de sus objetivos, únicamente nos centraremos en la capa física del estándar IEEE 802.16d, por lo que a continuación nos dispondremos a describirla un poco más en detalle. Dentro de la capa física, para la comprensión de los algoritmos a describir en hardware, únicamente será necesario conocer como está compuesta forma de onda de la señal transmitida y como se forman los preámbulos utilizados en las estimaciones. Como veremos más adelante, solo emplearemos para las estimaciones los preámbulos que se envían al comienzo de una trama. Por tanto, nos centraremos únicamente en estos puntos y comentaremos de la manera más breve posible el resto de aspectos a tener en cuenta de la capa física, los cuales no tiene demasiada importancia para los objetivos del proyecto. En los apartados que siguen, se ruega al lector que preste especial atención a los apartados de "*modulación de preámbulos*" y "*forma de onda en el dominio temporal y frecuencial*".

2.1 Capa física

La capa física que describiremos en breve hace referencia a la que hemos utilizado en el sistema de comunicaciones modelado. Concretamente, se basa en la modulación OFDM y está diseñada para un entorno NLOS en la banda de frecuencia de 2 a 11 GHz.

2.1.1 Forma de onda transmitida en el dominio temporal y frecuencial

Como ya vimos en el apartado de OFDM, la señal que se envía al canal es un símbolo OFDM junto con un prefijo cíclico necesario para mantener la ortogonalidad entre las subportadoras. Así, la forma de onda temporal de la señal transmitida es de la forma mostrada en la figura inferior, donde, según el estándar, T_b es el periodo de símbolo útil (calculado mediante la IDFT de los símbolos complejos) y T_g es el periodo del prefijo cíclico. Los T_g últimos microsegundos del periodo de símbolo útil se copian en la parte delantera, tal y

como se muestra en la figura (3.2), de manera que se tiene un nuevo periodo de símbolo T_s [38].

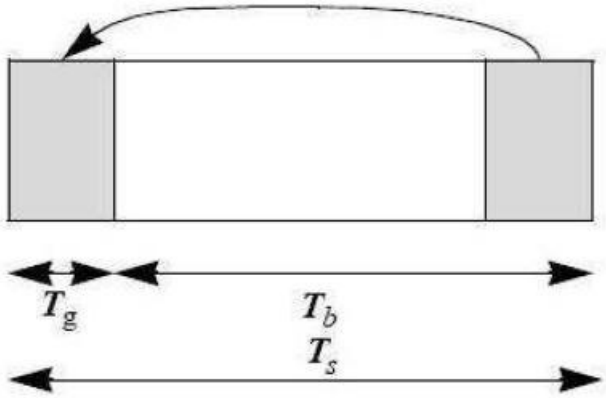


Figura 3.2: Forma de onda temporal [38]

T_g, también conocido como tiempo de guarda donde se inserta el prefijo cíclico, introduce cierta pérdida de la relación señal a ruido. Este parámetro podrá ser configurable por la BS dependiendo de las condiciones del canal, por lo que las SS deberán buscar los posibles valores del prefijo cíclico hasta dar con el correcto. En caso de que se cambie la longitud del prefijo cíclico, todas las SS deberán volverse a sincronizar. En el presente proyecto, a la hora de simular los algoritmos de estimación de canal y sincronización tiempo-frecuencia, se ha supuesto que el prefijo cíclico se mantiene constante tomando una longitud del mismo de 32 muestras.

Por otra parte, la respuesta en frecuencia del símbolo OFDM a transmitir se ilustra en la figura (3.3). En ella podemos observar cómo, de las 256 subportadoras, únicamente 200 subportadoras son utilizadas para la transmisión de datos. De las 56 subportadoras que restan, 8 son utilizadas para introducir tonos piloto, los cuales son empleados para realizar determinadas estimaciones. En el resto de subportadoras no se transmite información, y son empleadas en las bandas de guarda de los extremos del símbolo y en la frecuencia de DC.

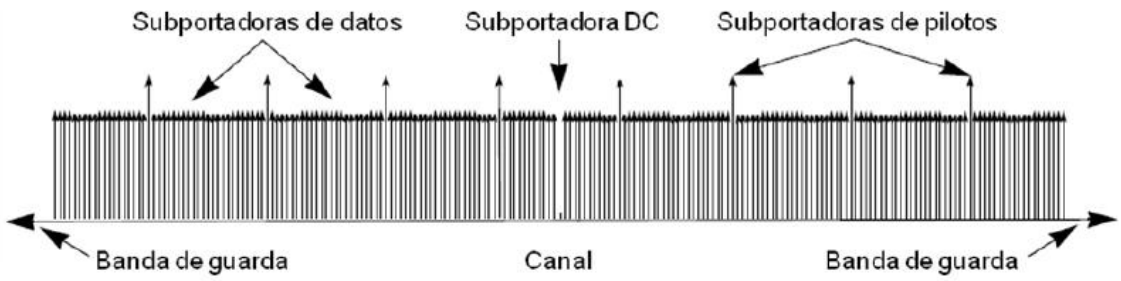


Figura 3.3: Respuesta en frecuencia de un símbolo OFDM [34]

En el estándar aparecen las 256 subportadoras del símbolo OFDM numeradas de la -127 a la 128, pero por cuestiones de comodidad en el presente proyecto se han numerado las subportadoras de la 0 a la 255. Así, la subportadora -128 se corresponderá con la 0, la subportadora -127 se corresponderá con la 1, y así sucesivamente hasta llegar a la subportadora 255.

La señal transmitida por el estándar IEEE 802.16d deberá cumplir con los siguientes parámetros [38]:

- Número de puntos de la FFT (DFT)→$N_{FFT} = 256$
- Número de subportadoras de datos → $N_{used} = 200$
- Factor de muestreo (n). Dependerá de ancho de banda del canal (Bandwidth, BW) según la tabla (3.1).

BW (MHz) múltiplo de:	n
1.75 ó otros BW	8/7
1.5	86/75
1.25	144/125
2.75	316/257
2	57/50

Tabla 3.1: Factor de muestreo

Este factor de muestreo, determinará con el BW y el número de subportadoras, la separación entre subportadoras, e indirectamente el periodo de símbolo útil. Se parte de la frecuencia de muestreo ($f_s = floor(n \cdot BW/8000) \cdot 8000$) para posteriormente calcular la separación

entre subportadoras como se vio en el apartado de OFDM ($\Delta f = f_s/N_{FFT}$).

- Relación entre tiempo de prefijo cíclico y periodo útil (G). En el presente proyecto se han usado 32 muestras por lo tanto esta relación es: $G = \frac{32}{256} = \frac{1}{8}$
  También, se puede elegir entre 1/4, 1/16, 1/32
- Número de portadoras de guarda superiores = 28
- Número de portadoras de guarda inferiores = 27

2.1.2 Codificación de canal

Para la codificación de canal, se aleatoriza la información aplicando la secuencia pseudo-aleatoria $1 + x^{14} + x^{15}$, se concatena un código Reed Solomon con un código convolucional (*Foward Error Correction*, FEC) y, por último, se realiza un entrelazado. Esto se realiza tanto en el *up-link* (UL) como en el *down-link* (DL) [38].

2.1.3 Modulación

- *Modulación de datos*
  Tras la codificación de canal los bits de datos se hacen corresponder con los símbolos de la constelación de a transmitir. El estándar soporta las siguientes modulaciones de datos: BPSK (*Binary Phase Shift Keying*), QPSK (*Quadrature Phase Shift Keying*), 16QAM (*16 Quadrature Amplitude Modulation*) y 64QAM [38]. En el sistema comunicaciones que se desea completar, se ha utilizado la modulación BPSK para los datos. Sin embargo, como veremos, el preámbulo (que es lo que realmente nos interesa en el presente proyecto) empleado para las funciones de estimación esta modulado en QPSK (figura (3.4)).

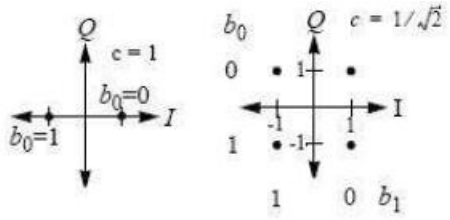


Figura 3.4: Constelaciones para BPSK y QPSK [34]

La energía de los símbolos de la constelación QPSK deberá estar normalizada a la unidad ($c = 1/\sqrt{2}$).

- *Modulación de pilotos*

  Los pilotos deberán ser transportados sobre las portadoras destinadas para tal fin. Cada piloto será modulado en función de la portadora a la que esté asociada. En el presente proyecto no utilizaremos los pilotos para realizar estimaciones, por lo no consideramos un aspecto importante para el desempeño del proyecto.

- *Modulación de preámbulos*

  Un preámbulo es una serie de muestras conocidas tanto en transmisión como en recepción, las cuales serán empleadas en el presente proyecto para realizar las funciones de estimación de canal y sincronización tiempo-frecuencia.

  Los preámbulos que el estándar utiliza, tanto en DL como en UL, derivan de la siguiente secuencia en el dominio de la frecuencia (P_{ALL}):

$P_{ALL}(-100:100)$ = {1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1+j, 1+j, 1+j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, 1+j, 1+j, -1+j, 1-j, -1+j, -1+j, 1-j, -1+j, 1-j, 1-j, 1+j, -1-j, -1-j, -1-j, 1+j, -1-j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1+j, -1+j, 1-j, -1-j, -1-j, -1-j, 1+j, 1-j, 1+j, 1+j, -1+j, 1-j, 1-j, 1-j, -1+j, 1-j, -1+j, -1+j, 1-j, -1+j, -1+j, -1+j, 1-j, -1+j, 1+j, 1+j, 1+j, -1-j, -1-j, -1-j, 1-j, 1+j, 1-j, -1+j, -1+j, -1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, 1-j, -1-j, -1-j, 1-j, -1+j, -1-j, -1-j, 1-j, -1+j, -1+j, -1+j, 1-j, -1+j, 1+j, 1+j, 1+j, -1-j, -1-j, -1-j, -1-j, 1+j, 1-j, 1-j}

Figura 3.5: Secuencia P_{ALL} [38]

La secuencia anterior, posee valores de la portadora -100 a la portadora 100. Sin embargo, notar que para las simulaciones realizadas en el presente proyecto se han tomado índices positivos para identificar a las subportadoras. También, es de destacar que los elementos de esta secuencia hacen referencia a símbolos de una constelación QPSK.

El estándar [38] permite dos tipos de preámbulos, un preámbulo conocido como preámbulo corto y otro conocido como preámbulo largo.

El preámbulo corto es un símbolo OFDM, el cual está formado por la concatenación de dos secuencias temporales de 128 muestras idénticas. A este símbolo OFDM, se le añade su correspondiente prefijo cíclico de igual manera que se hace en transmisión (de igual manera que hemos descrito en secciones anteriores). La secuencia de 128 muestras temporales se deriva de la siguiente secuencia en el dominio de la frecuencia (P_{EVEN}):

$$P_{EVEN} = \begin{cases} \sqrt{2}\, P_{ALL} & k_{mod2} = 0 \\ 0 & k_{mod2} \neq 0 \end{cases} \qquad (3.1)$$

En la figura (3.6) se ilustra la estructuración de este preámbulo.

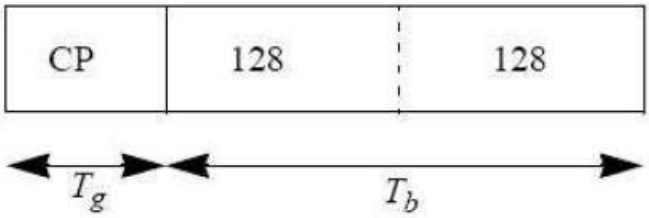


Figura 3.6: Preámbulo corto [34]

El preámbulo largo está constituido por dos símbolos OFDM. El primer símbolo OFDM está formado por la repetición 4 veces de una secuencia de 64 muestras temporales. A este símbolo OFDM, habrá que añadirle su correspondiente prefijo cíclico. La secuencia de 64 muestras temporales se deriva de la siguiente secuencia en el dominio de la frecuencia (P_{4x64}):

$$P_{4x64} = \begin{cases} \sqrt{2}\,\sqrt{2}\, conj(P_{ALL}) & k_{mod4} = 0 \\ 0 & k_{mod4} \neq 0 \end{cases} \qquad (3.2)$$

El segundo símbolo OFDM que forma el preámbulo largo es, simplemente, el preámbulo corto anteriormente descrito, tal y como podemos observar en la figura (3.7), la cual ilustra la estructura del preámbulo largo.

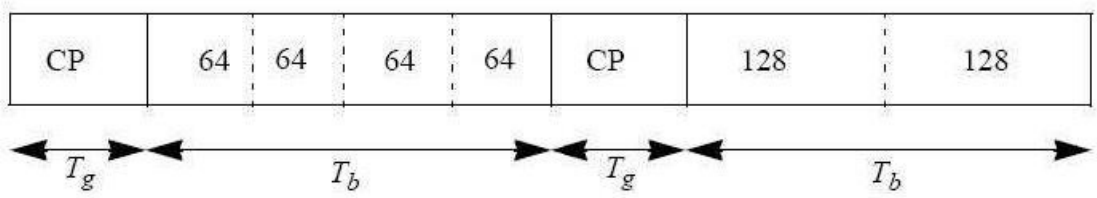


Figura 3.7: Preámbulo Largo [34]

A estos dos preámbulos, se le añade un tercer tipo de preámbulo para el caso en el que se emplee codificación espacio-tiempo en un sistema MIMO. El sistema de comunicaciones a completar, tiene este tipo de codificación para obtener ganancia por diversidad espacial, por lo tanto se utilizará este tipo de preámbulo. En el presente proyecto, estamos tratando con un sistema MIMO 2x2, por lo que el preámbulo empleado en la estimación de canal se transmite simultáneamente desde las dos antenas transmisoras y consistirá en un símbolo OFDM. La primera antena transmite un preámbulo que consiste en un símbolo OFDM con información en las subportadoras pares, la cual se deriva de la secuencia P_{EVEN}.Por otro lado, la segunda antena transmite, en el mismo instante de tiempo, un preámbulo que consiste en un símbolo OFDM con información en las portadoras impares ($k_{mod2} \neq 0$), la cual se deriva de la secuencia P_{ODD} que a continuación definimos como:

$$P_{ODD} = \begin{cases} \sqrt{2}\, P_{ALL} & k_{mod2} \neq 0 \\ 0 & k_{mod2} = 0 \end{cases} \qquad (3.3)$$

2.1.3.1 Preámbulo transmitido en el sistema de comunicación del presente proyecto.

El sistema de comunicación con el que trabajaremos en el presente proyecto, es un sistema MIMO 2x2 en el que se empleará codificación espacio-tiempo. Por tanto, para la estimación de canal y la estimación del desplazamiento frecuencial, utilizaremos el preámbulo anteriormente descrito, es decir, un preámbulo consistente en un símbolo OFDM en el que la información de las subportadoras pares es transmitida por una antena y la información de las subportadoras impares es transmitida por la otra.

Por otra parte, el algoritmo elegido para la estimación del desplazamiento temporal, está desarrollado para un sistema SISO-OFDM. Se parte de la base de que las antenas transmisoras comienzan a transmitir en el mismo instante de tiempo, por lo que se considera que, en recepción, el instante de tiempo en el que se recibe la trama transmitida, es el mismo en todas las antenas receptoras. Así, solo será necesario conocer el instante de tiempo en el que comienza la trama en una sola antena receptora, ya que en la otra antena receptora será el mismo tiempo. Por lo tanto, la sincronización temporal se realiza como si se tratase de un sistema SISO-OFDM. La sincronización temporal, se llevará a cabo mediante un símbolo OFDM, el cual se deriva de la repetición 4 veces de la secuencia P_{4x64} (igual que en el preámbulo largo).

La transmisión del preámbulo para el sistema de comunicación que estamos tratando, es una adaptación del preámbulo transmitido por el sistema MIMO-OFDM descrito en el artículo [2]. Según se muestra en la figura (3.8), en un primer instante solo la antena transmisora 1 emite el preámbulo anteriormente descrito para la sincronización temporal. A continuación, una vez el receptor se ha sincronizado temporalmente, las 2 antenas transmisoras emiten un símbolo OFDM que dará lugar al preámbulo para la estimación de canal y desplazamiento en frecuencia. Así, la antena 1 solo emite información en las portadoras pares (P_{EVEN}) y la antena 2 solo emite información en las portadoras impares (P_{ODD}), de manera que en una antena receptora, se tiene la suma de ambos símbolos OFDM, dando lugar a un único símbolo OFDM con información en todas las subportadoras.

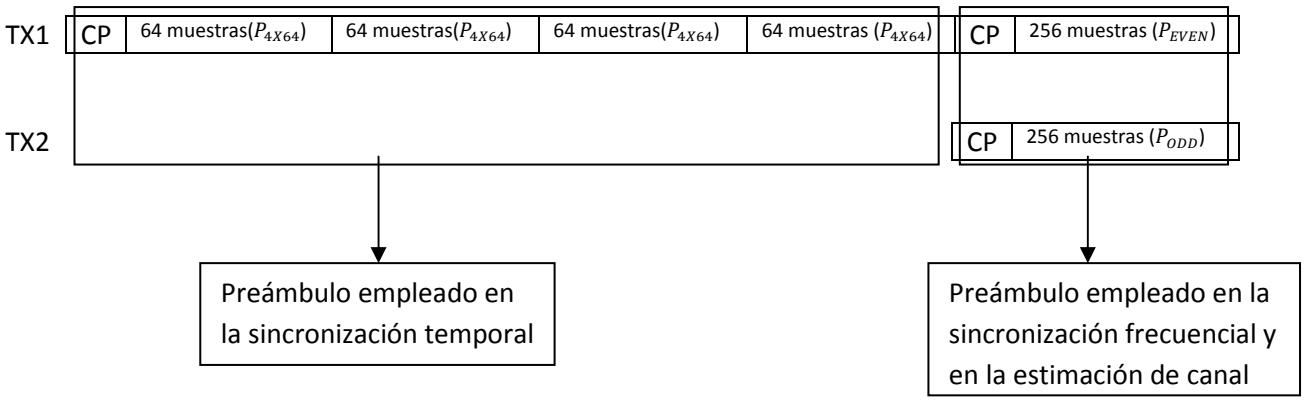


Figura 3.8: Preámbulo completo propuesto para el sistema de comunicaciones del presente proyecto

2.1.4 Estructuras de trama

Para el cumplimiento de los objetivos del proyecto, no es necesario el conocimiento de cómo está estructurada la trama de datos (campos y cabeceras), ya que únicamente utilizaremos los preámbulos transmitidos al comienzo de la trama. Simplemente, comentar en este apartado que existen diferentes tipos de tramas dependiendo de la topología de red [38]. Por ejemplo, si la topología de red es Punto-Multipunto (*Point-MultiPoint*, PMP) la trama toma una estructuración determinada, mientras que si ésta es tipo Mesh la trama de datos toma otra estructuración.

Capítulo 4

Plataforma empleada para el diseño de los módulos de estimación de canal y sincronicación tiempo-frecuencia

1. Introducción

Una vez contextualizado el proyecto mediante los capítulos anteriores, en el presente capítulo, se tratará caracterizar la plataforma empleada para el diseño de los módulos que desempeñan las funciones de estimación de canal y sincronización tiempo-frecuencia.

En una primera parte del capítulo, se tratará de describir cuales han sido las herramientas software empleadas, así como el hardware empleado para el diseño de los distintos módulos objetos de este proyecto. Por otro lado, en una segunda parte del capítulo, trataremos de describir el funcionamiento individualizado de cada uno de los elementos que intervienen en el modelado de los algoritmos a diseñar.

2. Herramientas empleadas

2.1 Matlab®

MATLAB® (*MATrix LABoratory*) es la herramienta software que se ha utilizado para simular el correcto funcionamiento de los algoritmos a diseñar en hardware. Se trata de un software desarrollado por "*The MathWorks Inc*®" que permite realizar cálculos con matrices. Esta herramienta incorpora un lenguaje de programación propio que goza de gran éxito entre la comunidad científica [17]. Este lenguaje de programación, es una magnífica, y sencilla, herramienta de alto nivel para desarrollar todo tipo de aplicaciones técnicas. Si se desea conocer más acerca de esta herramienta, se invita a lector que consulte el completo manual que se adjunta en la bibliografía [18].

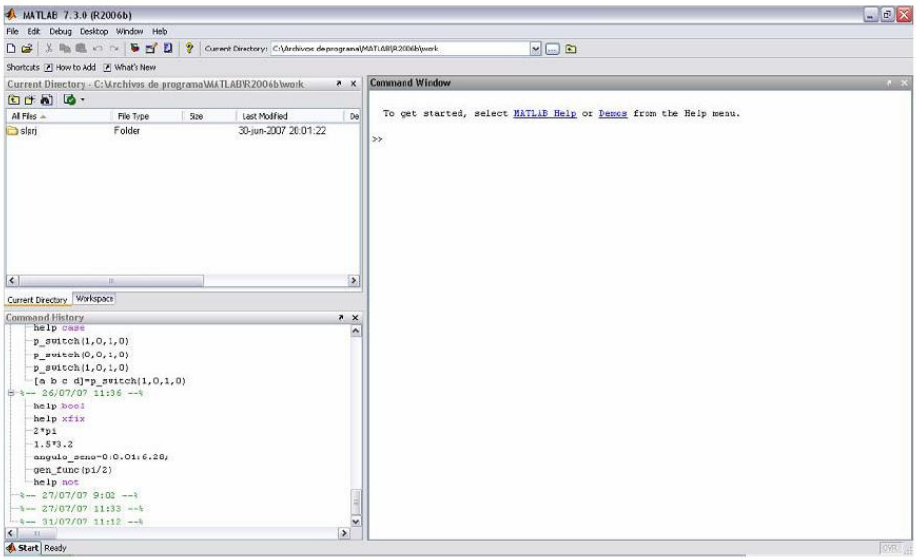


Figura 4.1: Entorno de trabajo de Matlab® [19]

2.2 Simulink ®(Simulation and Model-Based Desing)

Se trata de un entorno gráfico interactivo, el cual está integrado en Matlab®, que proporciona una serie de librerías de bloques que permiten simular, implementar y probar una serie de bloques de sistemas variables con el tiempo [20].

Las principales características de Simulink® las podemos resumir en:

- Gran numero de librerías de bloques predefinidos, las cuales permiten cierta flexibilidad, ya que es posible su ampliación. A este gran número de librerías, se le puede añadir nuevas librerías externas que hacen referencia a distintos campos de conocimiento (Tecnología aeroespacial, Sistemas de control, Procesamiento de señal, etc.).

- Editor gráfico interactivo donde crear los distintos modelos. Así, se permite administrar de una manera gráfica los distintos componentes de un sistema o modelo.

- Posibilidad de crear sistemas con distintos niveles de jerarquía, lo cual ayuda a una mejor organización y gestión de los mismos por parte del

diseñador (empleo de subsistemas). En la figura (4.2) podemos apreciar este concepto, además de la apariencia que presentan los modelos en Simulink®.

- Mediante "*Model explorer*" Simulink® permite navegar, crear, configurar y buscar todas las señales asociadas al modelo.

- Existe un interfaz de programación de aplicaciones (API) con el que Simulink® es capaz de comunicarse con otros programas de simulación o incorporar código externo.

- Simulink® permite la integración de ciertos algoritmos implementados en MATLAB® mediante el empleo de una serie de bloques "*Embedded MATLAB*".

- Distintos modos de simulación (normal, acelerador y acelerador rápido).

- Depurador y perfilador gráfico que permite ver los resultados de la simulación, así como comprobar el buen rendimiento y funcionamiento del modelo.

- Herramientas de análisis de los modelos que permiten detectar errores de modelado.

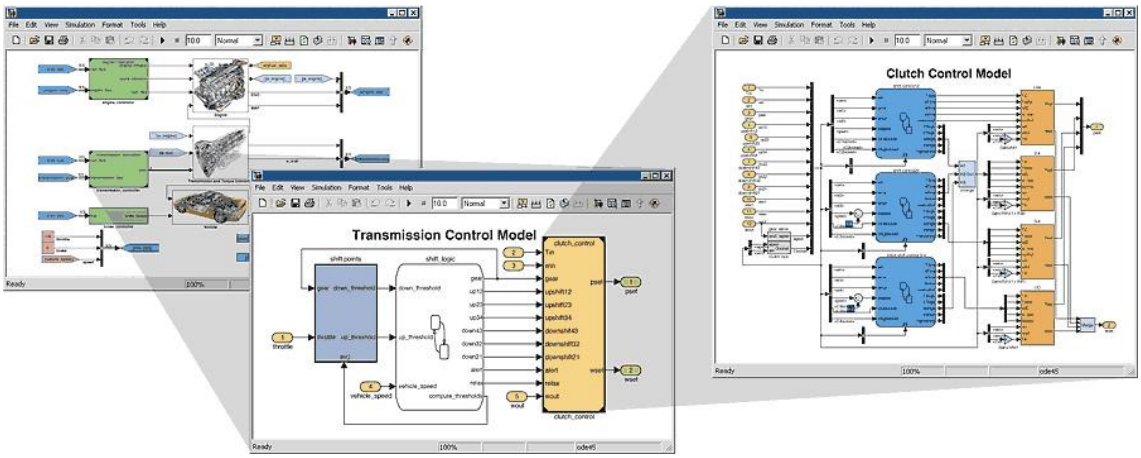


Figura 4.2: Entorno de trabajo de Simulink® [20]

2.3 FPGA (Field Programmable Gate Array)

Una FPGA es un dispositivo semiconductor constituido por una serie de bloques lógicos programables (CLBs, *Configurable Logic Blocs*) que se interconectan a través de una matriz de interconexión programable electrónicamente.

Al contrario de lo que sucede con los ASICs (*Circuito integrado para aplicaciones específicas*), las FPGAs son circuitos integrados que pueden utilizarse para diferentes aplicaciones. Es por ello, que una FPGA se caracteriza por tener mayor versatilidad y flexibilidad [19], ya que es un dispositivo abierto, es decir, se puede modificar para adaptarla a una posible nueva utilidad que el diseñador necesite. La mayoría de las FPGAs que se fabrican hoy en día, pueden reprogramarse para implementar un hardware determinado. Sin embargo, también existen FPGAs que sólo permiten una sola programación, debido al bajo coste de fabricación.

La figura (4.3) ilustra cómo está constituida una FPGA a grandes rasgos. En ella, distinguimos cuatro elementos básicos [21] [35]:

- **Bloques lógicos programables** (CLBs). Dependiendo del tipo de FPGA considerada, este elemento tendrá una arquitectura determinada. Por regla general, estará constituido por una tabla de funciones lógicas de 4 o 6 entradas (LUTs, *LookUp Tables*), biestables (*Flip-Flops*), multiplexores y lógica adicional, como por ejemplo podrían ser multiplicadores o sumadores. La LUTs se podrán configurar para implementar cualquier función lógica.

- **Matriz de interconexión**. Es la encargada de establecer la conexiones entre los distintos CLBs y los puertos de entrada/salida (IOBs, *Input Output Banks*)

- **Puertos de entrada/salida** (IOBs, *Input Output Banks*). Son los encargados de comunicar a la FPGA con el exterior y soportan diversos estándares de entrada/salida.

- **Dispositivo de gestión del reloj** (DCM, *Digital Clock Management*). Es el encargado de gestionar adecuadamente el reloj, de manera que llegue adecuadamente a todos los elementos síncronos del dispositivo.

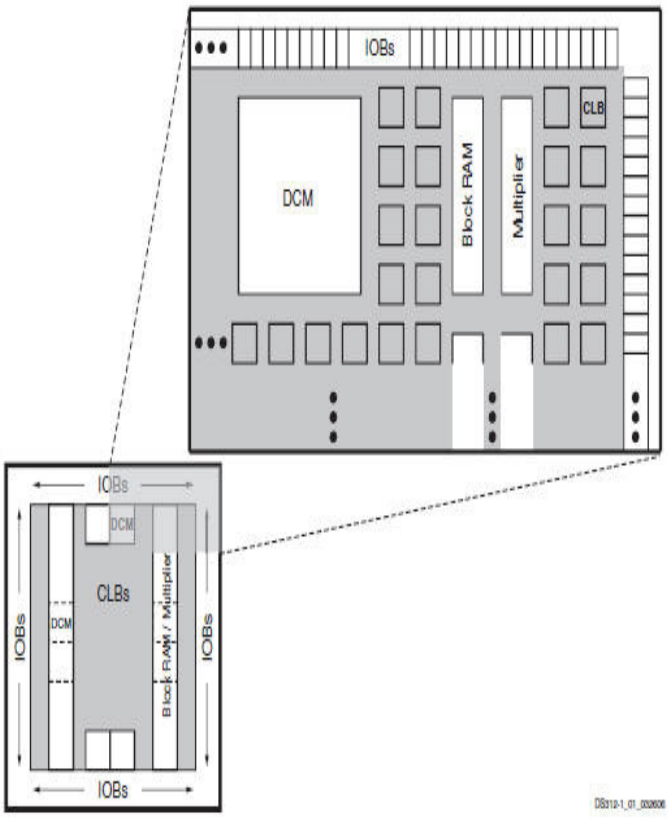


Figura 4.3: Elementos básicos de una FPGA [21]

En el presente proyecto, para describir en hardware los distintos módulos a diseñar, utilizaremos este tipo de dispositivo. Concretamente, como veremos a continuación, se emplea una FPGA de la familia Virtex-4 de Xilinx® que se haya integrada en una placa de adquisición de datos analógicos.

2.4 VHS-ADC/DAC Virtex-4 de Lrytech

La plataforma utilizada para la descripción del hardware a diseñar, es la placa de adquisición de datos analógicos VHS-ADC/DAC Virtex-4 desarrollada por la

empresa Lyrtech® (figura (4.4)) [35] [24], la cual integra una FPGA de la familia VIrtex-4. Este dispositivo es plenamente compatible con el software que emplearemos para el modelado del hardware a desarrollar, es decir, con *MATLAB®/Simulink®* y *System Generator For DSP®* (lo detallaremos más adelante).

Entre las características más importantes de la placa podemos destacar:

- Según las características técnicas del dispositivo, la placa es ideal para la implementación de sistemas de comunicación como el que se pretende diseñar, es decir, sistemas MIMO.
- Integración de una FPGA Virtex-4, la cual posee 152000 bloques lógicos y un consumo de potencia 2 veces inferior a otras familias de FPGAs.
- Frecuencia de muestreo configurable de desde 1 MHz hasta 105 MHz.
- Posibilidad de elegir entre el reloj integrado en la placa o uno externo.
- Memoria SDRAM (*Synchronous Dynamic Random Access Memory*) de 128 MB.
- Conexión al PC tipo Compact PCI del tipo 6U.
- 8 canales o conversores analógico digital con una velocidad máxima de muestreo de 105 megamuestras por segundo.
- Conexión al PC mediante el uso de una memoria flash y el puerto externo $I^2C/JTAG$.
- Kit de desarrollo software basado en C y VHDL.
- Posibilidad de diseñar hardware en un alto nivel de abstracción debido a la perfecta integración con *MATLAB®/Simulink®* y *System Generator For DSP®.*

Figura 4.4: Placa de adquisición de datos analógicos VHS-ADC/DAC Virtex-4 de Lrytech [35]

2.5 System Generator For DSP®

Para diseñar hardware sobre dispositivos como el descrito anteriormente, se suele utilizar lenguajes de descripción hardware (HDL, *Hardware Description Language*), como por ejemplo, VHDL (*VHSIC Hardware Description Language*) o Verilog. Sin embargo, existen herramientas de alto nivel que limitan el uso de códigos de descripción hardware en casos concretos [19].

Este tipo de herramientas implican un mayor nivel de abstracción y una mejor interacción con el diseñador. Este mayor nivel de abstracción hace que el diseñador trabaje en un nivel más alejado del hardware, lo que implica que este tipo de herramientas sean más fáciles de aprender y manejar. Diseñar a una mayor nivel de abstracción, supone un coste hardware, sin embargo, hoy en día se disponen de librerías con algunos bloques optimizados para determinadas FPGAs que obtienen resultados similares a los que se tienen empleando HDL directamente.

En el presente proyecto, se ha utilizado, para diseñar hardware, este tipo de herramienta, concretamente, se ha utilizado la herramienta *System Generator For DSP®.* Se trata de una herramienta software de alto nivel para el diseño de hardware, que se haya perfectamente integrada en Simulink®. *System Generator® (Xilinx Blockset, Xilinx Reference Blockset)* [21] aparece integrado en Simulink® como una librería externa del mismo, como podemos apreciar en la figura (4.5):

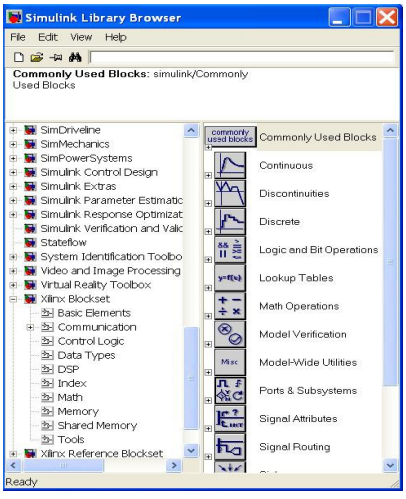


Figura 4.5: Librerías de System Generator® integradas en Simulink®

Así, será posible mediante el editor gráfico de Simulink®, crear modelos con los bloques prediseñados en las librerías proporcionadas por System Generator®. A partir del modelo generado en Simulink®, System Generator® podrá sintetizar el hardware deseado, previa generación del correspondiente código HDL que modela tal hardware [19].

Es de destacar una interesante ventaja que ofrece System Generator®, la co-simulación. Cuando se ejecuta una co-simulación, se comprueba el correcto funcionamiento del hardware implementado sobre la FPGA mediante Simulink®. En este caso, Simulink® actúa como interfaz a través del cual, se generan las señales de entrada al hardware implementado y se reciben las señales de salida del mismo para analizarlas posteriormente, es decir, Simulink® actúa como interfaz de comunicación con el hardware [19].

La versión de System Generator® utilizada en el presente proyecto es la 8.1.01, la cual soporta la siguiente lista de dispositivos: Virtex-5 LX, Virtex-5 LXT, Virtex-5 SXT, Virtex-5 FXT, Virtex-4 FX, Virtex-4 LX, Virtex-4 SX, Virtex-II Pro, Virtex-II, Virtex-E, Spartan-3A DSP, Spartan-3A, Spartan-AN, Spartan-3, Spartan-3E, Spartan-II y Spartan-IIE [35].

Para el diseño, simulación y descripción del hardware a diseñar, utilizaremos las librerías proporcionadas por System Generator®, concretamente, la librería *Xilinx Blockset y la librería Xilinx Reference Blockset*. En el siguiente apartado, trataremos de describir los bloques de estas librerías que hemos utilizado en los diseños a realizar.

3. Bloques empleados en los diseños

En el siguiente apartado, se tratará de describir todos los elementos que se han utilizado en el modelado de los algoritmos a diseñar. En un primer lugar, a nivel de simulación, será preciso introducir señales de entrada a nuestros diseños, por lo que para ello, será necesario la inclusión de bloques de la librería de Simulink, los cuales no implican un coste hardware a la hora de sintetizar el circuito diseñado. Así, en esta primera parte del apartado, se tratará de describir el funcionamiento de estos bloques Simulink®.

En un segundo lugar, se describirán los elementos empleados de las librerías de *System Generator For DSP® (Xilinx Blockset, Xilinx Reference Blockset)* [22] [23]

y de *Lytech* [24], entre los que distinguiremos aquellos que suponen un coste hardware y los que no.

3.1 Bloques de la librería Simulink

3.1.1 Puerto de entrada (Inport)

Figura 4.6: Puerto de entrada (Inport)

Se trata de una etiqueta (figura (4.6)) que se ha empleado para definir los puertos de entrada a los distintos subsistemas creados en los modelos realizados.

3.1.2 Puerto de salida (Outport)

Figura 4.7: Puerto de Salida (Outport)

Se trata de una etiqueta (figura (4.7)) que se ha empleado para definir los puertos de salida de los distintos subsistemas creados en los modelos realizados.

3.1.3 Subsistema (Subsystem)

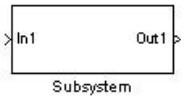


Figura 4.8: Subsistema (Subsystem)

Este bloque (figura (4.8)) se emplea para organizar mejor los diseños, de manera que podemos dividir un modelo siguiendo una jerarquía. Este bloque se utiliza para definir un subsistema dentro de un sistema. Al hacer doble click sobre el bloque, se abrirá otra ventana del editor gráfico de Simulink®, donde podremos añadir todos los elementos necesarios que compondrán el subsistema. Será necesario definir todas las entradas y salidas del subsistema mediante los puertos de salida y entrada anteriormente descritos.

3.1.4 From Workspace

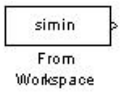


Figura 4.9: From Workspace

Este bloque (figura (4.9)) lee datos del "*Workspace*" de Matlab®. Si abrimos el bloque, observamos el cuadro que se ilustra en la figura (4.10), donde se definen los parámetros del mismo. En el campo "*data*", se indicará la expresión de una matriz de dos dimensiones, la cual leerá Simulink® del workspace de Matlab® en cada simulación. Una de las dos dimensiones indicará los instantes de tiempo en los que Simulink® leerá del "*Workspace*" y la otra los valores que se le pasan a Simulink® en cada instante de tiempo. Por ejemplo:

Siendo x la matriz de dos dimensiones requerida en el "*Workspace*" de Matlab®.

$$x = [1\ 2\ 3\ 4; 253\ 254\ 255\ 256];$$

Si introducimos x en el campo "*data*", la salida de este bloque irá cambiando en cada instante de tiempo de la siguiente manera:

Tiempo	1	2	3	4
Señal	253	254	255	256

Tabla 4.1: Salida del bloque From Workspace

Otra característica a destacar de este bloque, es la interpolación de los datos. Si seleccionamos "*interpole data*" en el cuadro de la figura (4.10), el bloque realizará una interpolación lineal si, en un instante de tiempo determinado, no se ha especificado que valor debe de sacar. Por ejemplo:

Si la estructura (matriz de 2 dimensiones) de datos es la siguiente:

Tiempo	1	2	3	4
Señal	253	254	¿	255

Tabla 4.2: Matriz de entrada

La salida del bloque será:

Tiempo	1	2	3	4
Señal	253	254	254.5	256

Tabla 4.3: Salida interpolada del bloque From Workspace

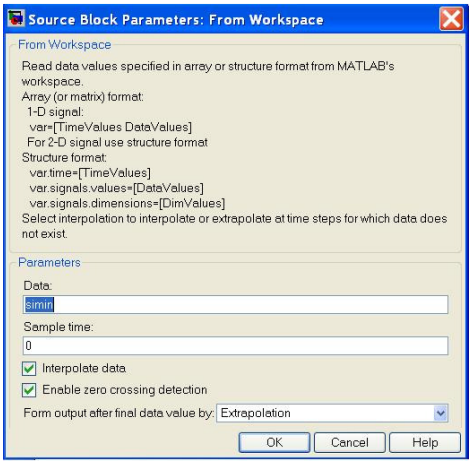


Figura 4.10 Cuadro de configuración del bloque "From Workspace"

En el presente proyecto, se simulará previamente el paso de la señal transmitida por el canal en Matlab®. Por ello, utilizaremos el bloque que

acabamos de describir, para importar los resultados de la simulación del canal, los cuales serán las señales de entrada a nuestro hardware.

3.1.5 To Workspace

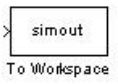


Figura 4.11: To Workspace

Este bloque (figura (4.11)) escribe su entrada en el "*Workspace*" de Matlab®. Si abrimos el bloque, observamos el cuadro que se ilustra en la figura (4.12), donde se definen los parámetros del mismo. En el campo "*Variable name*", determinamos el *Array* o estructura del workspace donde se escribirán los datos de entrada a este bloque. En el campo "*Save format*", se determina el formato del los datos que se escribirán en el "*Workspace*". Para el fin que emplearemos este bloque, solo nos interesa el formato *Array*, donde los datos que van llegando en cada instante de tiempo, se van escribiendo en un vector de una dimensión en el "*Workspace*". La posición de cada elemento de este vector, indicará el instante de tiempo en el que se ha escrito.

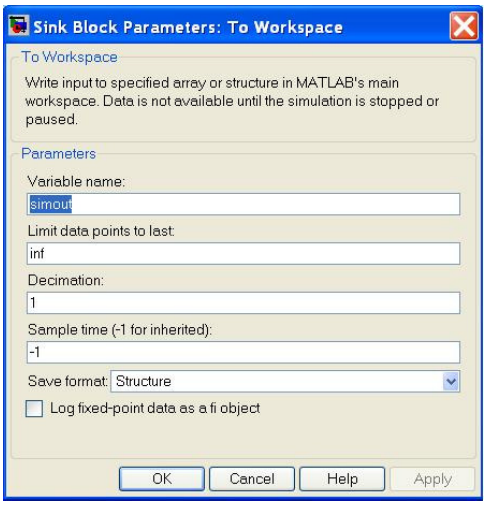


Figura 4.12: Cuadro de configuración del bloque "To Workspace"

Este bloque lo utilizaremos para importar al "*Workspace*" del Matlab® los resultados de las simulaciones en Simulink®. Los datos importados serán pasados a una función de Matlab® que mostrará los resultados de la simulación.

3.1.6 Escalón (Step)

Figura 4.13: Función escalón (Step)

Este bloque (figura (4.13)) genera como salida una función escalón. Si abrimos el bloque, observamos el cuadro que se ilustra en la figura (4.14), donde se definen los parámetros del mismo. Como podemos observar, se definen campos donde podemos indicar los valores que la señal puede tomar (*Initial value, Final value*) y el instante donde se produce el flanco de la señal (*Step time*).

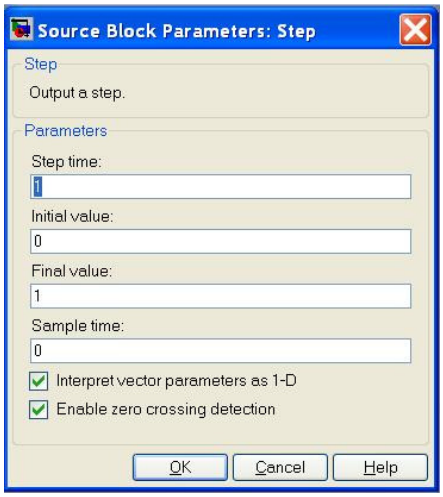


Figura 4.14: Cuadro de configuración del bloque Escalón (Step)

Este bloque será utilizado en los modelos diseñados para simular entradas de habilitación.

3.1.7 Constante (Constant)

Figura 4.15: Bloque Constante

Este bloque (figura (4.15)), simplemente genera un valor constante, el cual puede ser tanto real como complejo. Este bloque simplemente se ha utilizado para generar ciertas entradas en modelos de ejemplo, sin embargo, no ha sido utilizado en los diseños relativos al presente proyecto. Su cuadro de configuración se ilustra en la figura (4.16).

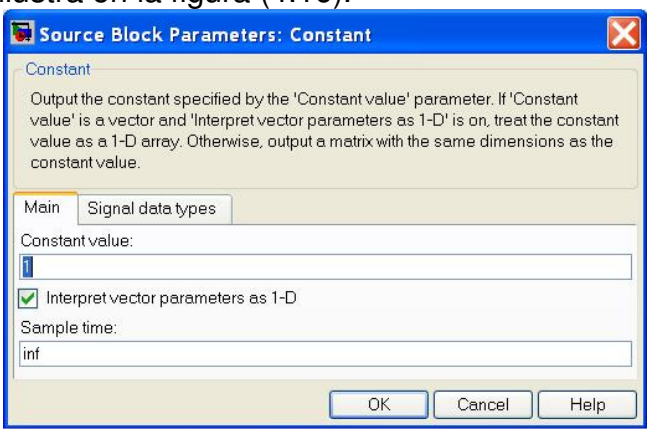


Figura 4.16: Cuadro de configuración del bloque Constante

3.1.8 Generador de pulsos (Pulse Generator)

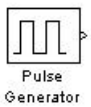


Figura 4.17: Bloque Generador de pulsos

Este bloque (figura (4.17)) simplemente genera un tren de pulsos rectangulares configurable. Si abrimos el bloque, observamos el cuadro que se ilustra en la figura (4.18), donde se definen los parámetros del mismo. Como podemos ver de la figura (4.18), los campos "*Amplitude*", "*Period*" y "*Pulse Width*", permiten configurar la amplitud, el periodo y la anchura del pulso de la señal que se

genera. Este bloque simplemente se ha utilizado para generar ciertas entradas en modelos de ejemplo, sin embargo, no ha sido utilizado en los diseños relativos al presente proyecto

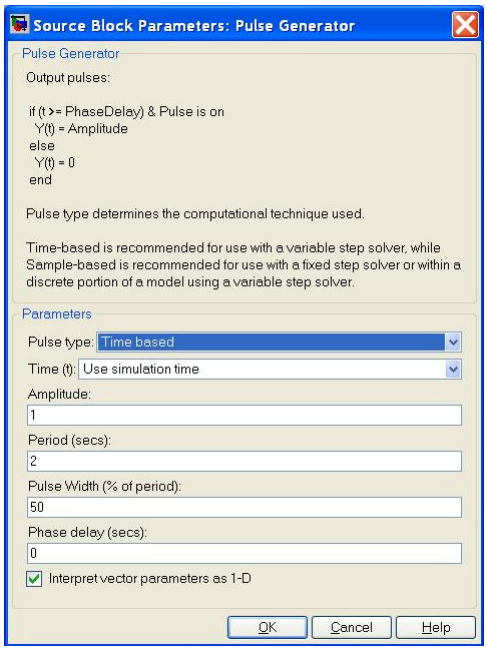


Figura 4.18: Cuadro de configuración del bloque Generador de pulsos

3.1.9 Terminación (Terminator)

Figura 4.19: Terminación (Terminator)

Este bloque (figura (4.19)) simula una terminación. Existen ciertos bloques empleados en los diseños, cuyas salidas no son utilizadas, por lo que se han conectado a una terminación.

3.2 Bloques sin coste Hardware

3.2.1 Bloques de la librería Xilinx Blockset

3.2.1.1 Bloque Xilinx System Generator

Cualquier modelo que incluya bloques de System Generator® debe incluir este bloque (figura (4.20)). Este bloque será el encargado de proporcionar el control del sistema y los parámetros de simulación del mismo. Además, ejecutará el generador de código y determinará como se deberá generar el código.

Figura 4.20: Bloque Xilinx System Generator

Al hacer doble click sobre el icono mostrado en la figura (4.20), obtenemos el cuadro en el que se detallan los parámetros del bloque. En la figura (4.21), se ilustra este cuadro, dentro del cual destacamos los siguientes campos:

- *Compilation*: determina el tipo de compilación que deberá llevar a cabo el generador de código (HDL Netlist, NGC Netlist, Bitstream, EDK Export Tool, Hardware Co-Simulation, Lyrtech® o Time Analysis).
- *Part*: determina la FPGA que se usará (Spartan2, Spartan2E, Spartan3, Spartan3E, Virtex, VirtexE, Virtex2, Virtex2P o Virtex4).
- *Target Directory*: identifica el directorio donde se guardan los ficheros generados tras la compilación.
- *Synthesis Tool*: determina la herramienta empleada en la sintetización del hardware. La herramienta de síntesis utilizada en el presente proyecto es Xilinx's XST®.
- *Hardware Description Languaje*: determina qué lenguaje HDL se empleará para la compilación. En nuestro caso, utilizaremos VHDL.
- *FPGA Clock Period*: indica el reloj del hardware. Deberá ser un valor entero de nanosegundos.

- *Clock Pin Location*: determina el pin del reloj. La información de localización del pin se guarda en un archivo (xcf o ngc) que es leído por las herramientas de implementación de Xilinx®. Además, en este archivo se incluyen otro tipo de restricciones.
- *Create Testbench*: cuando se activa esta casilla, se le indica a System Generator que cree un banco de pruebas.
- *Import as Configurable Subsystem*: si se encuentra activa, System Generator® crea un bloque como resultado de la compilación y un subsistema configurable.
- *Provide Clock Enable Clear Pin*: si se encuentra active, System Generator® crea un puerto para manejar la lógica de funcionamiento del reloj.
- *Override with doubles*: indica si todos los cálculos deben realizarse en doble precisión.
- *Simulink System Period*: determina el periodo en segundos utilizado en Simulink®.
- *Block Icon Display*: determina que información se mostrará en los iconos de los distintos bloques del modelo tras la compilación del mismo. Se proporciona un menú desplegable donde podemos elegir la información a mostrar. Se puede elegir entre: Default (información de bloque por defecto), Sample Rates (muestra la tasa de datos en cada puerto), Pipeline Stages (muestra la latencia de los puertos de entrada), HDL Port Names (muestra los nombres de los puertos), Input data types (muestra los tipos de dato de las señales de entrada) y Output data types (muestra los tipos de dato de las señales de salida)

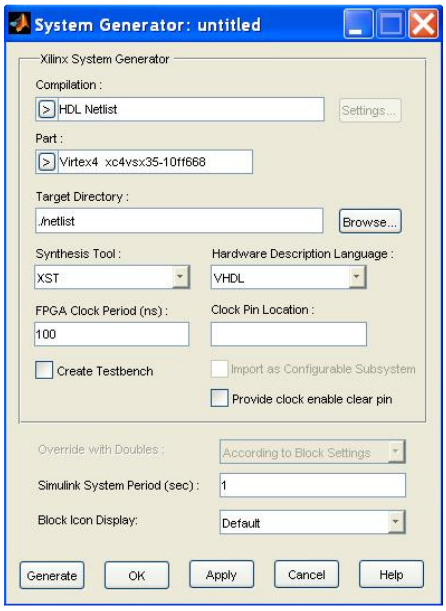


Figura 4.21: Cuadro de configuración del bloque Xilinx System Generator

3.2.1.2 Bloque Xilinx Wavescope

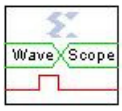


Figura 4.22: Bloque Xilinx Wavescope

Este bloque (figura (4.22)) sirve de herramienta de ayuda al diseñador para analizar y depurar los diseños. Al hacer doble click sobre el icono del bloque, aparece una ventana donde se pueden visualizar todas las señales de un modelo (figura (4.23)). Al término de una simulación, este bloque permite ver todas las señales implicadas en el modelo, ya sean analógicas o digitales. En caso de que se esté tratando con señales digitales, la herramienta permite elegir el formato en el que se desean mostrar (binario, decimal, hexadecimal, etc.)

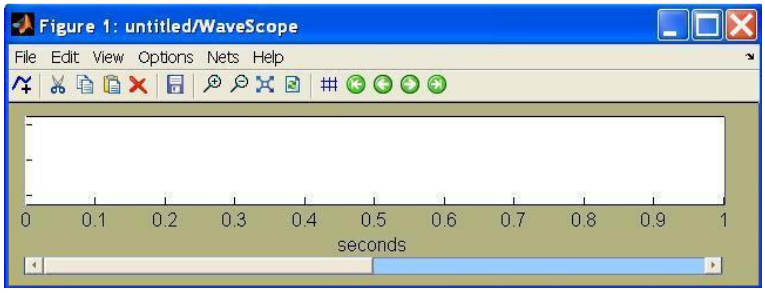


Figura 4.23: Ventana de visualización Wavescope

Existe una forma sencilla de añadir señales a este visualizador. En primer lugar, se deberá abrir el bloque haciendo doble click sobre el icono que lo representa. Una vez abierto el visualizador, se deberán seleccionar presionado la tecla "*shift*" los elementos del modelo cuyas señales se desean visualizar. Una vez seleccionados los elementos, se pulsa la tecla "*Add Selected Nets*" para añadir las señales a la ventana de visualización. Cada señal añadida podrá ser editada, es decir, se podrá cambiar el color, nombre, formato, etc.

Cuando se tienen todas las señales a visualizar añadidas, se simula el modelo y aparecen representados en la ventana de visualización los valores que toman las señales a analizar.

3.2.2 Bloques de la librería Lrytech VHS-ADAC Blockset

3.2.2.1 Bloque VHS-ADAC Board Configuration

Este bloque (figura (4.24)) está destinado a configurar la FPGA sobre la cual sintetizaremos nuestro hardware a través de System Generator®.

Figura 4.24: Bloque VHS-ADAC Board Configuration

Al hacer doble click sobre el icono mostrado en la figura (4.24), obtenemos el cuadro en el que se detallan los parámetros del bloque. En la figura (4.25), se ilustra este cuadro, dentro del cual destacamos los siguientes campos:

- *Clock Type*: determina que reloj se usará en el diseño. Se podrá elegir entre, *Single Step* (tipo de reloj empleado en co-simulación) y *Free Running* (usado para el resto de tareas).
- *VHS Board Type*: determina el tipo de placa.
- *Copy bitstream to current directory*: en caso de que esté activada, el fichero *.bit utilizado para programar la FPGA se guarda en el directorio actual de trabajo ("*current directory*" en el entorno de trabajo de Matlab®).

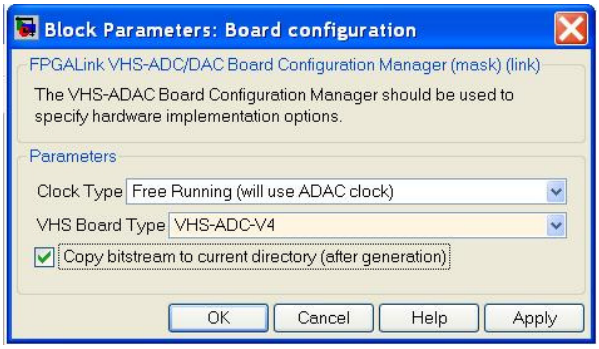


Figura 4.25: Cuadro de configuración del bloque VHS-ADAC Board Configuration

3.3 Bloques con coste Hardware

3.3.1 Bloques de la librería Xilinx Blockset

3.3.1.1 Bloque Xilinx Gateway In

Figura 4.26: Bloque Xilinx Gateway In

Este bloque (figura (4.26)) se utiliza para definir una entrada de nuestro diseño hardware. Convierte los tipos de datos enteros, dobles y de punto fijo disponibles en Simulink® a tipos de datos de punto fijo para System Generator®. Este bloque define y nombra a un puerto de entrada en el HDL que System Generator® genera.

Si abrimos el bloque, observamos el cuadro que se ilustra en la figura (4.27), donde se definen los parámetros del mismo. En los diseños realizados únicamente ha sido modificado el campo "*Output Type*", donde se define el tipo de dato para la señal de salida de este bloque, dejando los demás campos por defecto.

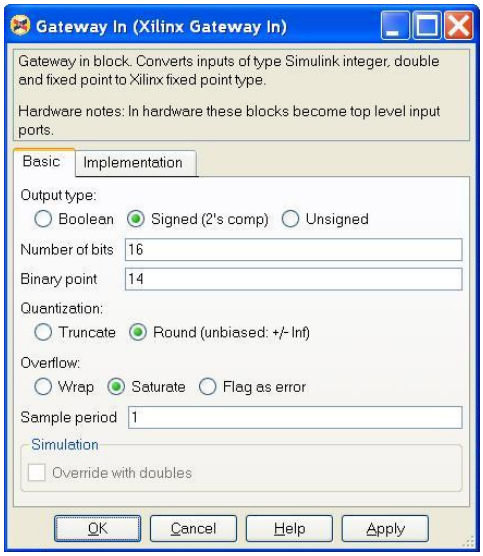


Figura 4.27: Cuadro de configuración del Bloque Xilinx Gateway In

3.3.1.2 Bloque Xilinx Gateway Out

Figura 4.28: Bloque Xilinx Gateway Out

Este bloque (figura (4.28)) se utiliza para definir una salida de nuestro diseño hardware. Convierte los tipos de datos de punto fijo a doble. Este bloque define y nombra a un puerto de salida en el HDL que System Generator® genera.

3.3.1.3 Bloque Xilinx Delay

Figura 4.29: Bloque Xilinx Delay

Este bloque (figura (4.29)) retarda un determinado número de ciclos de reloj la señal de entrada. En concreto, es un registro de desplazamiento, de longitud configurable, que provoca el retardo.

- *Puertos del bloque*
  1) Puerto de entrada: señal sobre la que se aplica el retardo.
  2) Puerto de salida: señal de entrada retardada.

- *Configuración básica*
  Si hacemos doble click sobre el icono de la figura (4.29), obtenemos el cuadro de configuración del bloque (figura (4.30)).

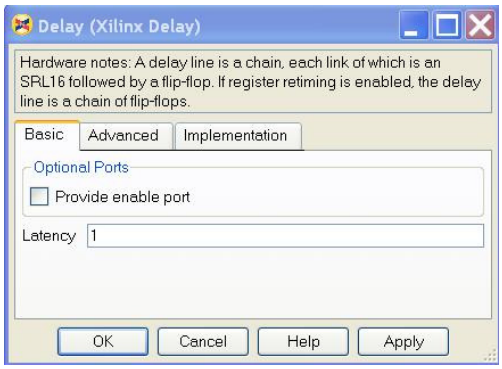


Figura 4.30: Cuadro de configuración del Bloque Xilinx Delay

Del cuadro de configuración, es de destacar el campo "*latency*" donde se introducirá el número de ciclos de reloj que deseamos retardar la señal de entrada.

- *Ejemplo de utilización*
  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.31)):

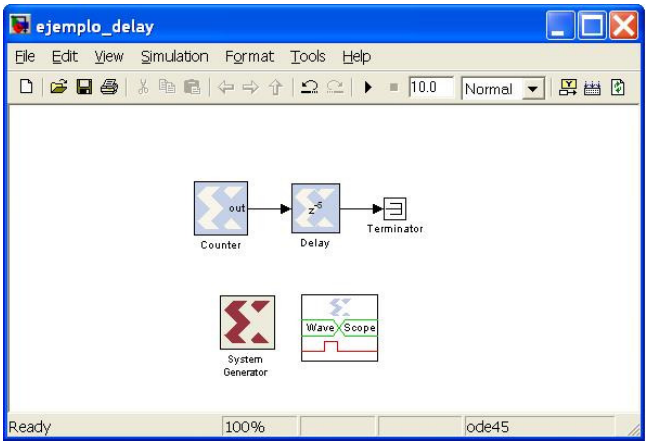


Figura 4.31: Ejemplo de utilización del Bloque Xilinx Delay

En este modelo se utiliza un bloque contador (*Counter*) que servirá como fuente de entrada y un bloque "***Delay***" que retardará la señal de entrada 5 ciclos de reloj (ver cuadro de configuración de la figura (4.32)). Notar que en el modelo también aparecen otros elementos que ya comentamos en apartados anteriores (*Wavescope y System Generator®*).

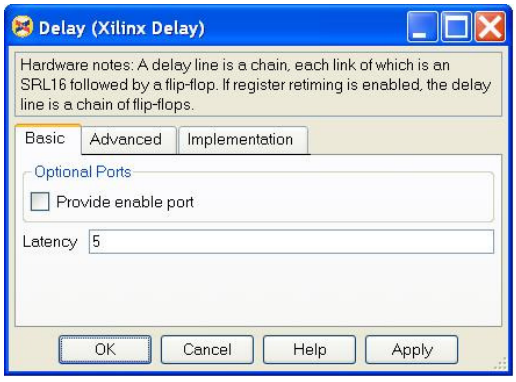


Figura 4.32: Cuadro de configuración del Bloque Xilinx Delay para el modelo de ejemplo

De la simulación de este modelo, el bloque "***Wavescope***" nos devuelve representadas las señales de entrada y de salida del bloque "***Delay***". Esto se muestra en la figura (4.33), donde podemos observar el retardo de 5 ciclos de reloj.

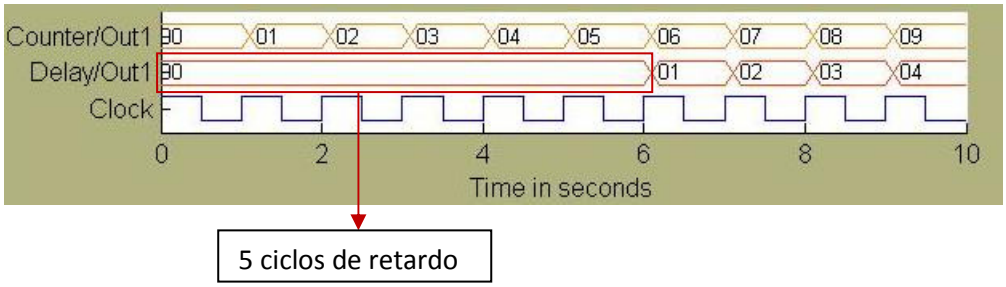


Figura 4.33: Resultado de la simulación para el modelo de ejemplo

3.3.1.4 Bloque Xilinx Shift

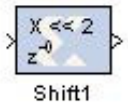


Figura 4.34: Bloque Xilinx Shift

Este bloque (figura (4.34)) realiza un desplazamiento lógico a la derecha o izquierda de los datos de entrada. La salida de este bloque mantiene el punto binario en la misma posición que la entrada.

- *Puertos del bloque*
  1) Puerto de entrada: señal sobre la que se aplica el desplazamiento.
  2) Puerto de salida: señal de entrada con los bits desplazados según el sentido de desplazamiento indicado.

- *Configuración básica*
  Si hacemos doble click sobre el icono de la figura (4.34), obtenemos el cuadro de configuración del bloque (figura (4.35)).

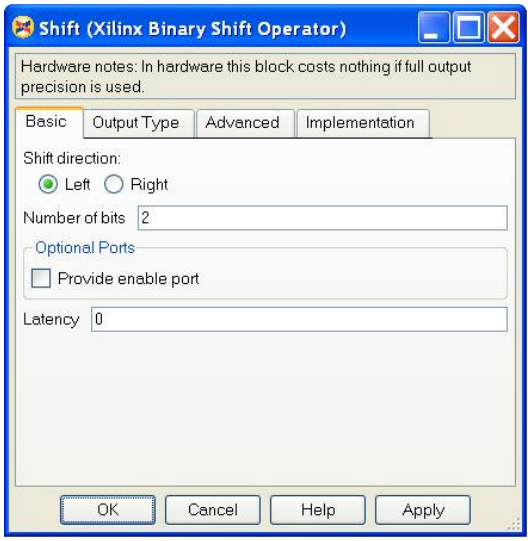


Figura 4.35: Cuadro de configuración del Bloque Xilinx Shift

En el cuadro de configuración mostrado en la figura (4.35), destacamos dos campos, *Shift direction y Number of bits*. *Shift direction* determina el sentido de desplazamiento, de manera que si se quiere desplazar a la derecha, se introducen ceros por la parte derecha de la palabra de entrada y si se quiere desplazar a la izquierda, se introducen ceros por la parte izquierda. *Number of bits* determina el número de bits que se desean desplazar.

Es posible configurar los tipos de datos de salida de este bloque mediante la pestaña del cuadro de configuración "*Output type*".

- *Ejemplo de utilización*

  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.36)):

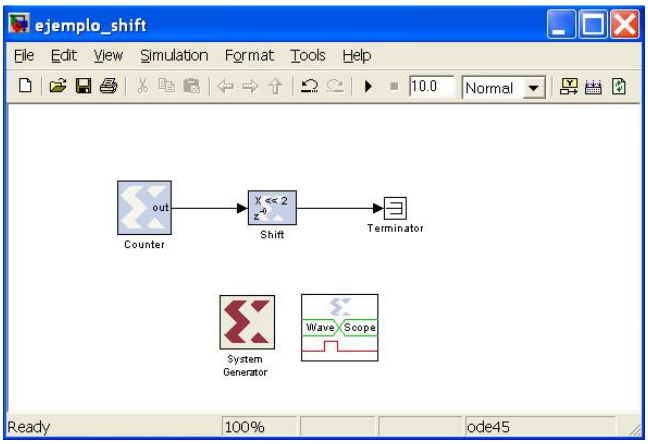


Figura 4.36: Ejemplo de utilización del Bloque Xilinx Shift

En este modelo se utiliza un bloque contador que servirá como fuente de entrada y un bloque "***Shift***" que desplazará los bits de entrada dos posiciones a la izquierda mediante la inclusión de ceros (ver cuadro de configuración de la figura (4.35)). Notar que en el modelo también aparecen otros elementos que ya comentamos en apartados anteriores (*Wavescope y System Generator®*).

De la simulación de este modelo, el bloque "***Wavescope"*** nos devuelve representadas las señales de entrada y de salida del bloque "***shift***". Esto se muestra en la figura (4.37), donde podemos observar que los 8 bits de datos a la entrada se desplazan dos posiciones a la izquierda. Se aprecia cómo se insertan dos ceros en los bits menos significativos, de manera que se desplazan los bits a la izquierda.

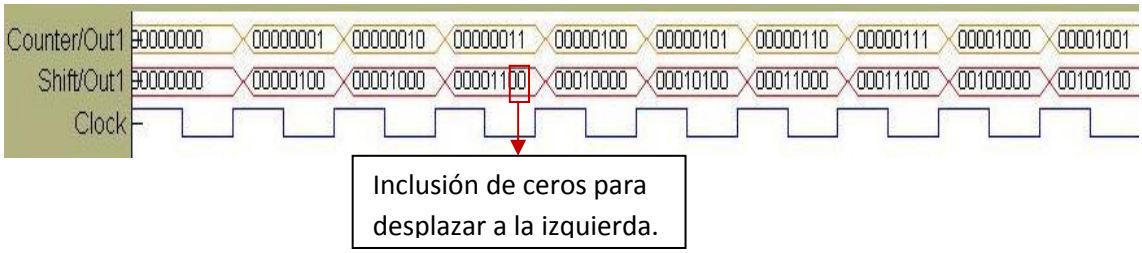


Figura 4.37: Resultado de la simulación para el modelo de ejemplo

3.3.1.5 Bloque Xilinx Constant

Figura 4.38: Bloque Xilinx Constant

Este bloque (figura (4.38)) genera un valor constante que puede ser con signo, sin signo o de tipo booleano. También, posibilita la creación de instrucciones mediante el bloque DSP48.

- *Puertos del bloque*
  1) Puerto de salida: señal constante generada.

- *Configuración básica*
  Si hacemos doble click sobre el icono de la figura (4.38), obtenemos el cuadro de configuración del bloque (figura (4.39)).

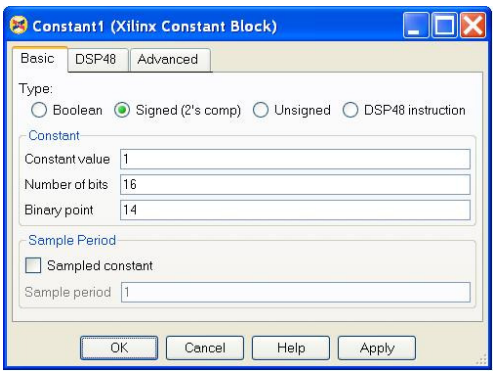


Figura 4.39: Cuadro de configuración del Bloque Xilinx Constant

En el cuadro de configuración mostrado en la figura (4.39), destacamos los campos: *type, constant value, number of bits, binary point, sampled constant. Type* determina el tipo de constante, *constant value* determina el valor de la constante, *number of bits* determina el número de bits que representarán la constante si el tipo elegido es punto fijo, *binary point* determina la posición del punto fijo y *sampled constant* asocia un periodo de muestreo a la constante.

- *Ejemplo de utilización*

  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.40)):

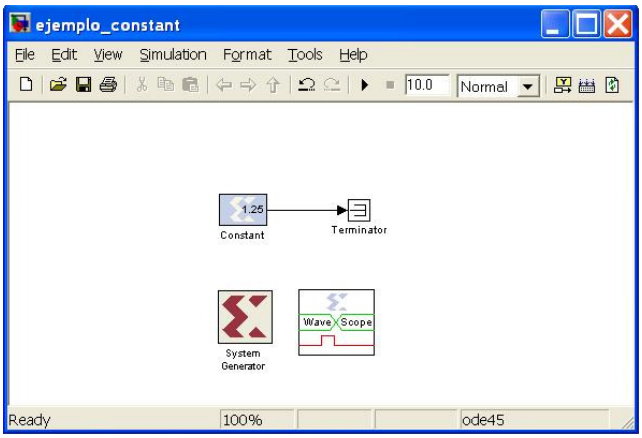


Figura 4.40: Ejemplo de utilización del Bloque Xilinx Constant

En este modelo, únicamente representaremos la salida de un bloque "***Constant***" configurado para obtener el valor binario 1.01, es decir, 1.25 en decimal (ver cuadro de configuración de la figura (4.41)). Notar que en el modelo también aparecen otros elementos que ya comentamos en apartados anteriores (*Wavescope y System Generator®*).

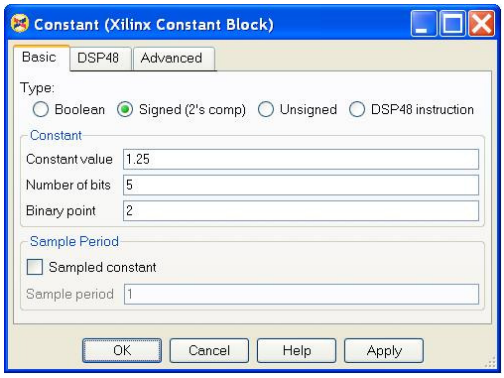


Figura 4.41: Cuadro de configuración del Bloque Xilinx Constant para el modelo de ejemplo

De la simulación de este modelo, el bloque "***Wavescope***" nos devuelve representada la señal de salida del bloque "***Constant***". Esto se muestra en la figura (4.42), donde podemos observar la constante esperada, es decir, 001.01 (1.25 en decimal).

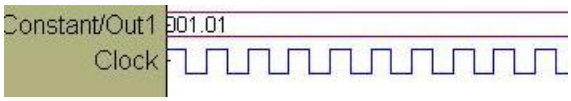


Figura 4.42: Resultado de la simulación para el modelo de ejemplo

3.3.1.6 Bloque Xilinx CMult

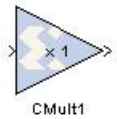


Figura 4.43: Bloque Xilinx CMult

Este bloque (figura (4.43)) amplifica la señal en su puerto de entrada, de manera que la salida es la señal de entrada multiplicanda por una constante.

- *Puertos del bloque*
  1) Puerto de entrada: señal a multiplicar.
  2) Puerto de salida: señal de entrada multiplicada por la constante.

- *Configuración básica*
  Si hacemos doble click sobre el icono de la figura (4.43), obtenemos el cuadro de configuración del bloque (figura (4.44)).

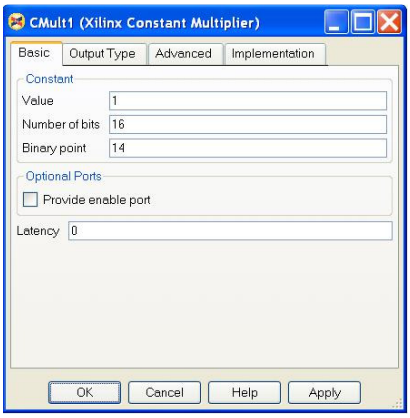


Figura 4.44: Cuadro de configuración del Bloque Xilinx CMult

En el cuadro de configuración mostrado en la figura (4.44), destacamos los campos: *Value, Number of bits y Binary Point. Value* determina el valor de la constante, *Number of Bits* el número de bits para representar la constante y *Binary Point* la localización del punto binario de la constante.

Es posible configurar los tipos de datos de salida de este bloque mediante la pestaña del cuadro de configuración "*Output type*".

- *Ejemplo de utilización*
  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.45)):

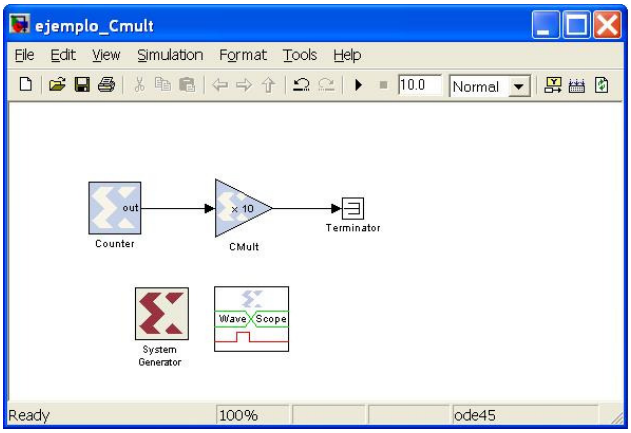


Figura 4.45: Ejemplo de utilización del Bloque Xilinx CMult

En este modelo se utiliza un bloque contador (*Counter*) que servirá como fuente de entrada y un bloque "***CMult***" que multiplicará por 10 los valores de entrada generados por el contador (ver cuadro de configuración de la figura (4.46)). Notar que en el modelo también aparecen otros elementos que ya comentamos en apartados anteriores (*Wavescope y System Generator®*).

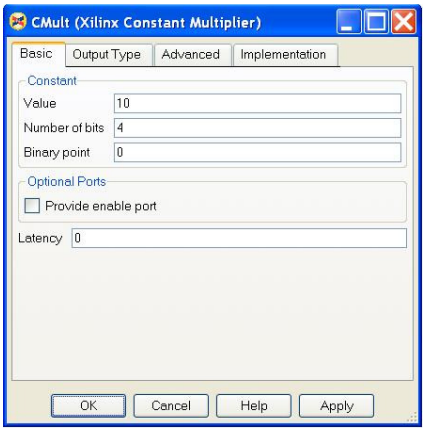


Figura 4.46: Cuadro de configuración del Bloque Xilinx CMult para el modelo de ejemplo

De la simulación de este modelo, el bloque "***Wavescope***" nos devuelve representadas las señales de entrada y de salida del bloque "***CMult***". Tal y como podemos observar en la figura (4.47), los valores del contador se hayan multiplicados por 10 a las salida del bloque "***CMult***".

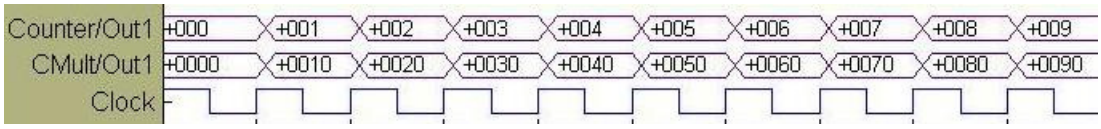


Figura 4.47: Resultado de la simulación para el modelo de ejemplo

3.3.1.7 Bloque Xilinx Mux

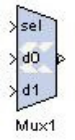


Figura 4.48: Bloque Xilinx Mux

Este bloque (figura (4.48)) implementa un multiplexor. Consta de un número de puertos de entrada configurable y una entrada para seleccionar los puertos de entrada.

- *Puertos del bloque*
  1) Puertos de entrada: todo el conjunto de señales de entrada que se desean seleccionar a la salida. Se pueden configurar hasta un máximo de 32 puertos de entrada.
  2) Entrada de selección (*sel*): señal que indica qué entrada se seleccionará. Los bits de este puerto deberán ser de tipo "sin signo" (*unsigned*) y tener un número de bits adecuado al número de puertos de entrada.
  3) Puerto de salida: señal de entrada seleccionada por el multiplexor.

- *Configuración básica*
  Si hacemos doble click sobre el icono de la figura (4.48), obtenemos el cuadro de configuración del bloque (figura (4.49)).

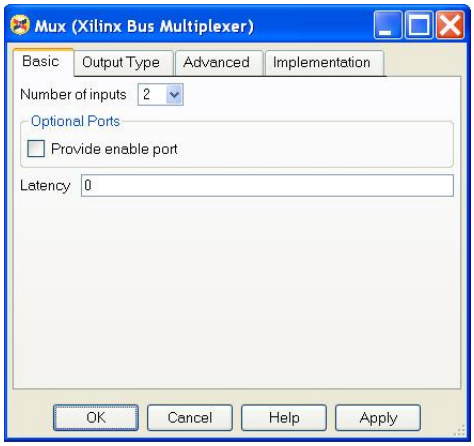


Figura 4.49: Cuadro de configuración del Bloque Xilinx Mux

Del cuadro de configuración, es de destacar el campo "*Number of Inputs*" donde se seleccionará el número de puertos de entrada.

Es posible configurar los tipos de datos de salida de este bloque mediante la pestaña del cuadro de configuración "*Output type*".

- *Ejemplo de utilización*

Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.50)):

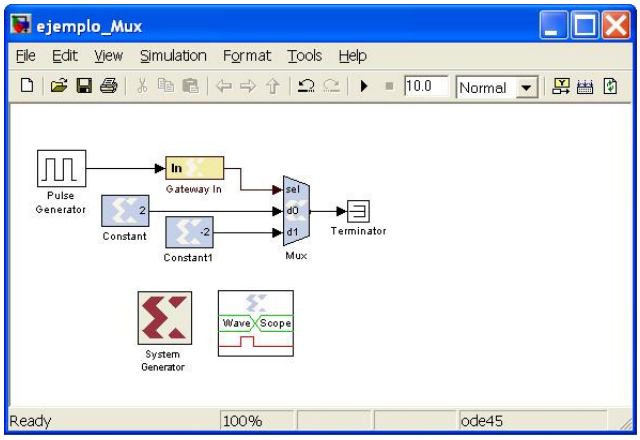


Figura 4.50: Ejemplo de utilización del Bloque Xilinx Mux

En este modelo se pretende mostrar el funcionamiento de un multiplexor de dos entradas. Como entradas al multiplexor, tenemos dos constantes con los valores decimales 2 y -2 y una señal de selección que variará de 0 a 1 para elegir entre los dos puertos de entrada. La señal de selección, es generada mediante un bloque de la librería Simulink (*Pulse Generator*), configurado para generar una señal con un periodo de dos segundos y un "*duty clicle*" del 50%. El multiplexor queda configurado como se muestra en la figura (4.49).Notar que en el modelo también aparecen otros elementos que ya comentamos en apartados anteriores (*Wavescope y System Generator®*).

De la simulación de este modelo, el bloque "***Wavescope***" nos devuelve representadas las señales de entrada y de salida del bloque "***Mux***". Tal y como podemos observar en la figura (4.51), cada vez que la señal de selección está a 0 seleccionamos la primera entrada (2 en decimal) y cada vez que está a 1 seleccionamos la segunda entrada (-2 en decimal).

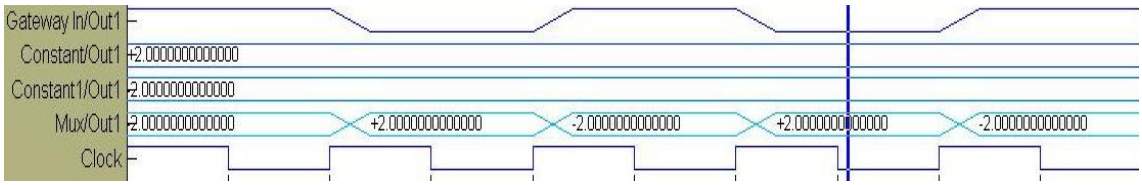


Figura 4.51: Resultado de la simulación para el modelo de ejemplo

3.3.1.8 Bloque Xilinx ROM (Read-Only Memory)

Figura 4.52: Bloque Xilinx ROM

Este bloque (figura (4.52)) implementa una memoria ROM. Los valores almacenados en la ROM tienen el mismo tipo, número de bits y posición de punto binario. A cada valor almacenado se le hace corresponder con una dirección leída en el puerto de entrada. La dirección de entrada, que debe de ser de tipo "sin signo" (*unsigned*), tendrá un número de bits acorde al número de valores almacenados en la memoria.

- *Puertos del bloque*
  1) Puerto de entrada (*addr*): señal que determina qué dirección se debe leer. Es un entero de tipo "sin signo" (*unsigned*).
  2) Puerto de salida: valor leído de la memoria ROM.

- *Configuración básica*
  Si hacemos doble click sobre el icono de la figura (4.52), obtenemos el cuadro de configuración del bloque (figura (4.54)).

  En el cuadro de configuración mostrado en la figura (4.54), destacamos los campos: *Depth, Initial value vector y Memory Type. Depth* indica el número de valores almacenado en la ROM, *Initial value* vector determina los valores que se almacenarán en la ROM a través de la definición de un vector de valores y *Memory Type* determina qué tipo de memoria se implementa (*distributed RAM o Block RAM*).

  Es posible configurar los tipos de datos de salida de este bloque mediante la pestaña del cuadro de configuración "*Output type*".

- *Ejemplo de utilización*
  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.53)):

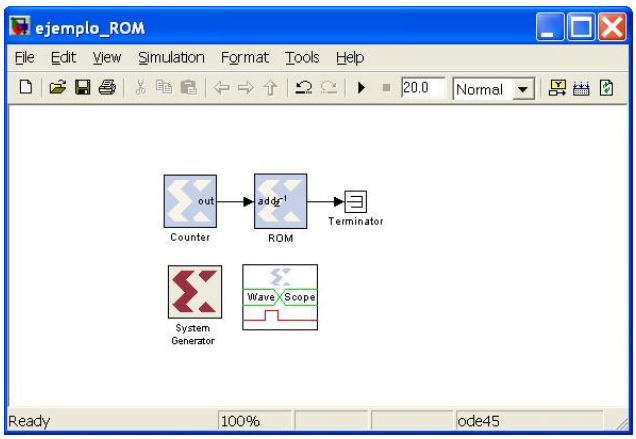


Figura 4.53: Ejemplo de utilización del Bloque Xilinx ROM

En este modelo se utiliza un bloque contador para generar las direcciones que se leerán y un bloque "***ROM***" configurado para almacenar 16 valores que van desde el 16 al 32 (ver cuadro de configuración de la figura (4.54)). El contador contará del 0 al 15, de manera que se irán leyendo secuencialmente los valores almacenados en las direcciones que van del 0 al 15. Notar que en el modelo también aparecen otros elementos que ya comentamos en apartados anteriores (*Wavescope y System Generator®*).

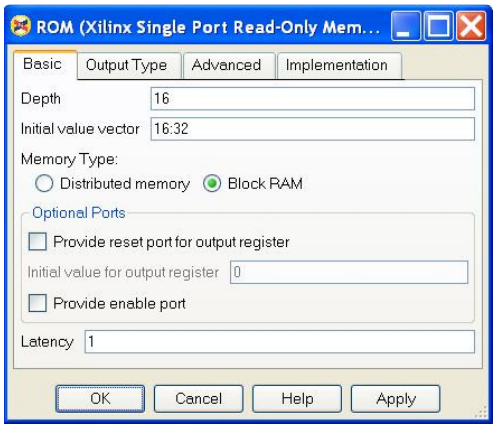


Figura 4.54: Cuadro de configuración del Bloque Xilinx ROM para el modelo de ejemplo

De la simulación de este modelo, el bloque "***Wavescope***" nos devuelve representadas las señales de entrada y de salida del bloque "***ROM***", las cuales se corresponderán con la dirección a leer y el dato leído, respectivamente. En la figura (4.55), podemos observar que se tarda

necesariamente un ciclo de reloj en leer un dato de la ROM (latencia de un ciclo). Esto es algo característico del bloque y como mínimo siempre tendremos un ciclo de reloj de latencia. Si nos fijamos en el primer dato de la ROM, es decir, el 16 (dirección +00), éste se tiene en el puerto de salida, un ciclo después de leer su dirección de memoria.

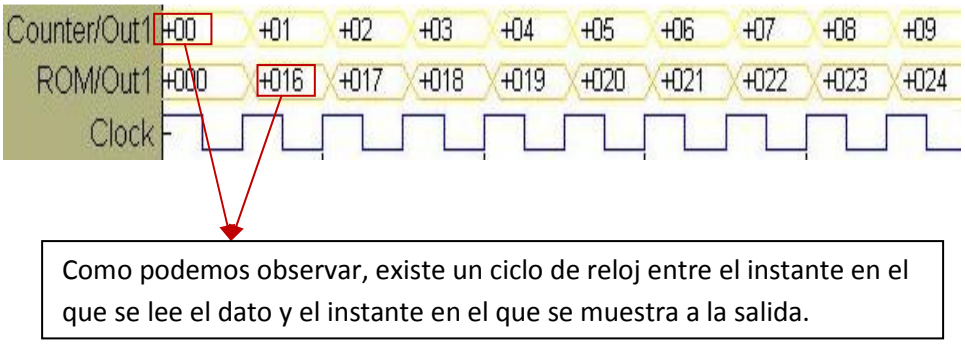


Figura 4.55: Resultado de la simulación para el modelo de ejemplo

3.3.1.9 Bloque Xilinx AddSub

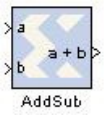


Figura 4.56: Bloque Xilinx AddSub

Este bloque (figura (4.56)) implementa un sumador o un restador. Se puede configurar de una de las dos maneras, o incluso, se puede configurar para que se pueda elegir el instante en que actúa como sumador y el instante en el que actúa como restador. Sin embargo, esta última opción no la emplearemos nunca en el presente proyecto.

- *Puertos del bloque*
  1) Puertos de entrada (a, b): son las dos señales que deseamos sumar.
  2) Puerto de salida ($a + b$, $a - b$): señal resultado de la suma o resta de las dos entradas.

- *Configuración básica*

  Si hacemos doble click sobre el icono de la figura (4.56), obtenemos el cuadro de configuración del bloque (figura (4.57)).

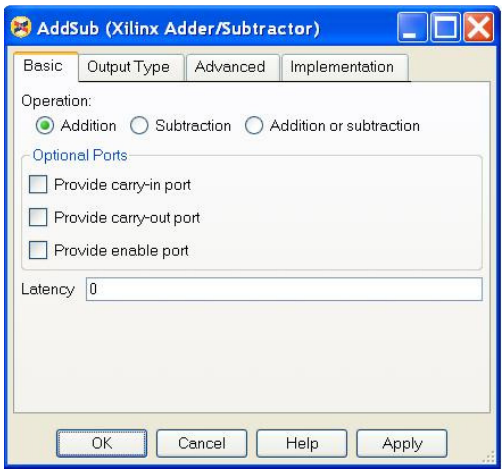


Figura 4.57: Cuadro de configuración del Bloque Xilinx AddSub

En el cuadro de configuración mostrado en la figura (4.57), solo destacaremos el campo "*Operation*", con el cual seleccionaremos el tipo de operación que realizará el bloque. Los puertos opcionales que aparecen en el cuadro de configuración no los emplearemos.

Por otra parte, será posible configurar los tipos de datos de salida de este bloque mediante la pestaña del cuadro de configuración "*Output type*".

- *Ejemplo de utilización*

  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.58)):

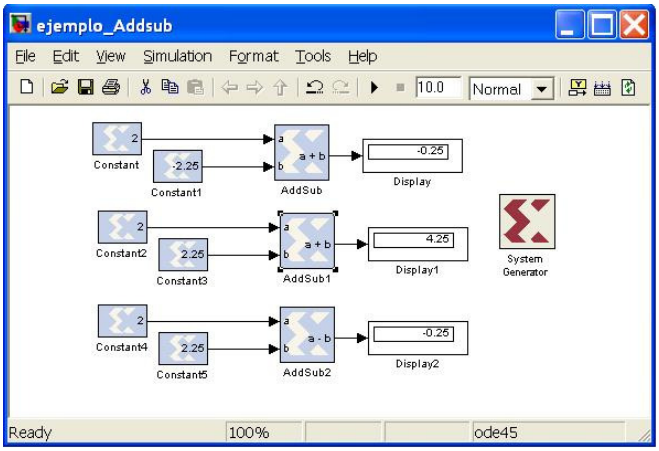


Figura 4.58: Ejemplo de utilización del Bloque Xilinx AddSub

En este modelo se pretende mostrar el funcionamiento del bloque "***AddSub***". En él, podemos observar los dos primeros bloques configurados como sumadores, y el bloque restante configurado como restador. Como entradas se han escogido bloques "***Constant***", configurados con los valores que se pueden ver sobre los iconos de dichos bloques. Para representar los resultados de las sumas y restas del modelo, se ha empleado un bloque de la librería de Simulink (bloque "***display***"). Como podemos apreciar en la figura, los resultados reflejados en los "*displays*" son correctos. Notar que en el modelo también aparece el bloque "***System Generator***", el cual siempre debe aparecer en los modelos en los que se usen bloques de System Generator®.

3.3.1.10 Bloque Xilinx Black Box

Figura 4.59: Bloque Xilinx Black Box

Este bloque (figura (4.59)) permite incluir código VHDL o Verilog en el modelo Simulink® que se esté creando. El código asociado al bloque debe cumplir una serie de restricciones:

- El identificador de la entidad no puede coincidir ni con el nombre del modelo Simulink® ni con el de entidades reservadas a System Generator®.

- No se permiten puertos bidireccionales y deben de ser de tipo *std_logic_vector.*

- Siempre deben aparecer los puertos de reloj y habilitación de reloj. Obligatoriamente, deben nombrarse "*clk*" y "*ce*". Estos puertos deben de ser de tipo *std_logic.*

Al añadir el bloque al modelo Simulink®, se abre una ventana que permite seleccionar el código VHDL o Verilog que se asociará al bloque y se ejecuta una herramienta de configuración que genera un archivo de configuración mediante una función de Matlab® que gestionará los puertos de entrada y salida, relojes, y archivos relacionados con el bloque. Esta función de Matlab® se podrá modificar con el editor de texto de Matlab® según las necesidades que se requieran (en la referencia bibliográfica [8] podemos consultar como configurar estas función). En cuadro de configuración del bloque "***Black Box***" tiene el aspecto que se muestra en la figura (4.60), donde destacamos el campo "*Block configuration m-function*" donde se indica la función de configuración asociada al bloque. El nombre de la función de configuración siempre aparece como **"(nombre_del_archivo.*vhd)_config",** por lo que será sencilla la identificación del fichero .*vhd que tiene asociado el bloque.

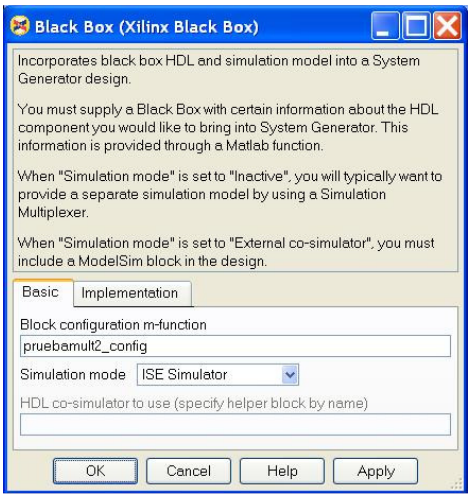


Figura 4.60: Cuadro de configuración del Bloque Xilinx Black Box

- *Ejemplo de utilización*

Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.61)):

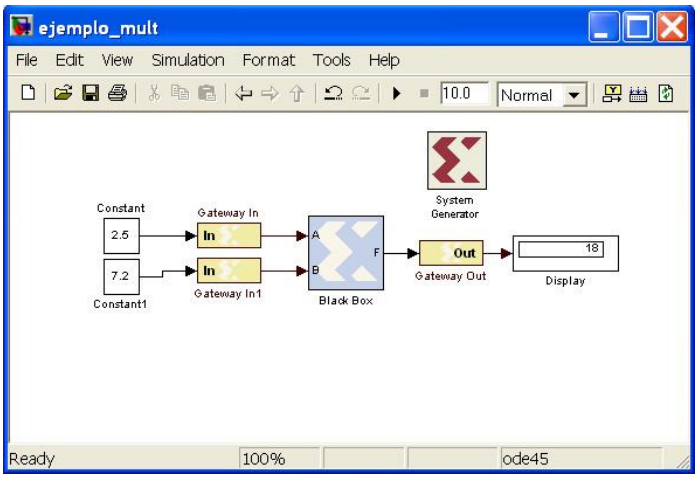


Figura 4.61: Ejemplo de utilización del Bloque Xilinx Black Box

En este modelo se ha modelado un multiplicador mediante un bloque "***Black Box***". Como podemos ver en la figura (4.61), el valor mostrado en el "*display*" se corresponde con el producto de los dos valores de entrada. El cuadro de configuración del bloque "***Black Box***" se corresponde con el de la figura (4.60). El código VHDL asociado a este bloque "***Black Box***" se puede encontrar en el anexo B. Es fácil deducir cuál de los códigos que aparecen en el anexo B es el correspondiente al bloque que estamos tratando, si consultamos el nombre de la función de configuración en el cuadro de configuración de la figura (4.60).

3.3.1.11 Bloque Convert

Figura 4.62: Bloque "Convert"

Este bloque (figura (4.62)), simplemente convierte los datos de entrada al tipo de datos deseado. Por ejemplo, se pueden convertir los datos de entrada a tipo "*signed*" o "*unsigned*". Un ejemplo de utilización de este bloque, lo podremos

observar en el siguiente punto (figura (4.65)) donde será empleado para adaptar los datos de entrada a un bloque "***FFT v3_1***".

3.3.1.12 Bloque Xilinx FFT v3_1

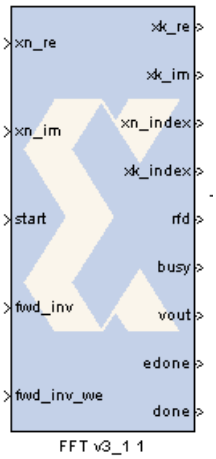


Figura 4.63: Bloque FFT v3_1

Este bloque (figura (4.63)) realiza la DFT (*Discrete Fourier Transform*) mediante el algoritmo *Cooley-Tukey*.

- *Puertos del bloque*
  Este bloque se puede configurar bajo distintas implementaciones (ver figura (4.64)). En el presente proyecto utilizaremos la implementación "*Pipelined Streaming I/O*". Con esta implementación destacamos los siguientes puertos:
  - Puertos de entrada:
    - *xn_re*: es la parte real de los datos de entrada. Deben de ser tipo "signed" con un número de bits S=8, 12, 16, 20 o 24 y con el punto binario en S-1.
    - *xn_im*: es la parte imaginaria de los datos de entrada. Deben de ser tipo "signed" con un número de bits S=8, 12, 16, 20 o 24 y con el punto binario en S-1.
    - *Start*: señal que indica el comienzo de una trama. Cuando se produce un flaco de subida en esta entrada se indica al bloque que comience a hacer la FFT o IFFT (*Fast Fourier Transform o Iverse Fast Fourier Transform*). Esta entrada debe de ser de tipo booleana.

- *Fwd_inv*: señal de entrada que indica si el bloque realiza una FFT o IFFT. Si esta en nivel alto realiza una FFT y si esta en nivel bajo realiza una IFFT. Esta entrada debe de ser de tipo booleana.
- *Fwd_inv_we*: señal utilizada para cargar el valor de la entrada "*fwd_inv*". Si está en nivel alto se carga el valor de "*fwd_inv*" que indica el tipo de transformada a realizar. Esta entrada debe de ser de tipo booleana.

- Puertos de salida:
  - *Xk_re*: señal de salida para la parte real. Bajo este tipo de implementación, se produce una extensión de la parte entera de los datos de salida. La parte entera crece en un número de bits $(1 + \log_2 N)$, donde N es el número de puntos de la FFT o IFFT.
  - *Xk_im*: señal de salida para la parte imaginaria. Bajo este tipo de implementación, se produce una extensión de la parte entera de los datos de salida. La parte entera crece en un número de bits $(1 + \log_2 N)$, donde N es el número de puntos de la FFT o IFFT.
  - Los puertos "*xn_index, xk_index, rfd, busy, vout, edone, done*", no les daremos uso en el presente proyecto, por tanto no son de importancia en este apartado. Si se desea más información puede consultar la referencia bibliográfica [4].

- *Configuración básica*

  Si hacemos doble clic sobre el icono de la figura (4.63), obtenemos el cuadro de configuración del bloque (figura (4.64)).

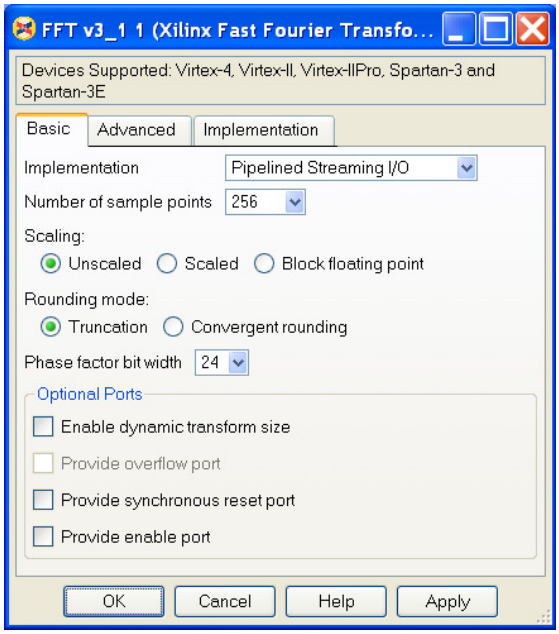


Figura 4.64: Cuadro de configuración del bloque "FFT v3_1"

En el cuadro de configuración de la figura destacamos los campos: "*implementation*" para seleccionar el tipo de implementación, "*Scaling*" donde se selecciona si se desea escalado y "*Number of sample points*" donde se selecciona el número de puntos de la FFT o IFFT.

- *Ejemplo de utilización*

  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink (figura (4.65)):

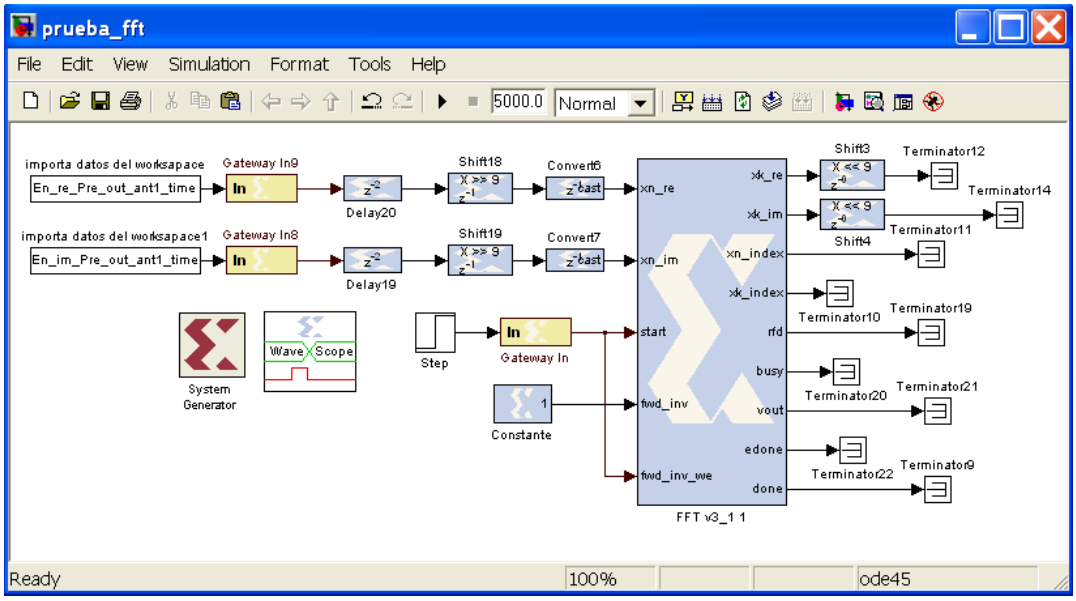


Figura 4.65: Ejemplo de utilización del bloque FFT v3_1

Como este bloque se va a tener que emplear para hacer la FFT sobre el preámbulo de la trama del sistema de comunicación considerado en el presente proyecto, se ha decidido poner como ejemplo la FFT del preámbulo utilizado en una antena. Así, en la figura (4.65), se ilustra el modelo que realiza esta tarea. Es de destacar, sobre la figura (4.65), las siguientes consideraciones:

- El bloque está configurado para realizar al FFT mediante la conexión de la entrada "*fwd_inv*" a un bloque "***Constant***" configurado para generar un estado alto su salida.
- La entrada "*Start*" está conectada a una entrada de habilitación (bloque "**step**") que determina cuando el bloque "***FFT v3_1***" debe comenzar a capturar la trama de entrada para comenzar a realizar la FFT.
- Los datos de entrada son de tipo "*signed*" de 24 bits con el punto binario en el bit 14, por lo que será necesario adaptar los datos de entrada para tener el tipo de dato soportado por este bloque. Para ello, utilizamos un bloque "***Shift***" para realizar un desplazamiento lógico a la derecha y poner el punto binario en el penúltimo bit más significativo. El bloque "***Shift***" amplía el número de bits de salida para evitar que, al desplazar los bits, se eliminen bits útiles, por lo que es necesario truncar los bits menos significativos de la salida del bloque "***Shift***" para adaptar los datos al tamaño

requerido por el bloque "***FFT v3_1***". Para ello, empleamos bloques "***Convert***" configurados para sacar datos tipo "*signed*" de 24 bits con punto binario en el bit 23. Ver cuadros de configuración de las figuras (4.66), (4.67) y (4.68).

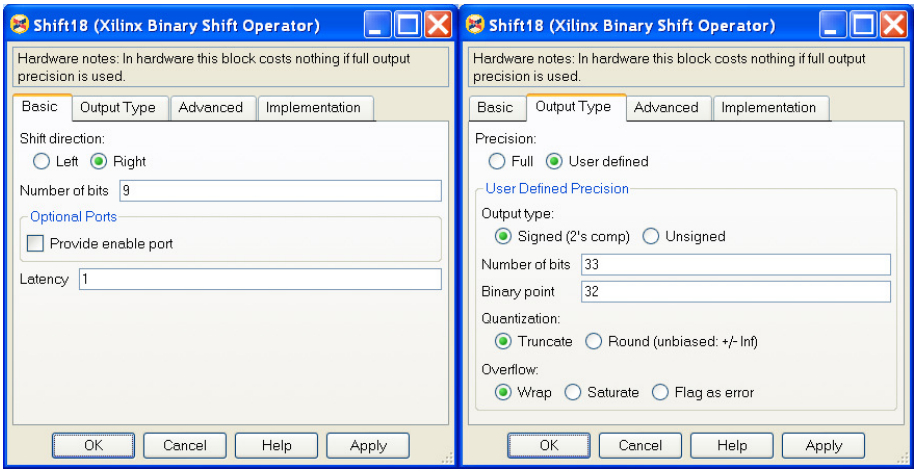


Figura 4.66: Cuadro de configuración de los bloques "Shift" de entrada

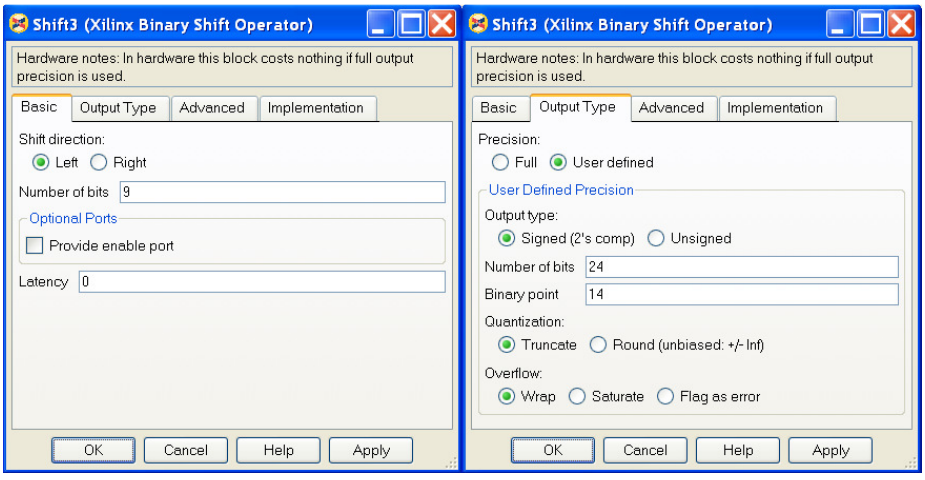


Figura 4.67: Cuadro de configuración de los bloques "Shift" de salida

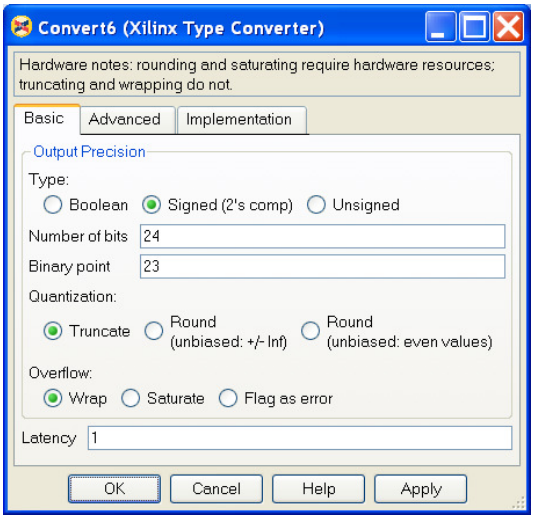


Figura 4.68: Cuadro de configuración de los bloques "Convert"

- A la salida habrá que adaptar los datos al formato que se tenía a la entrada. Esto se realiza, mediante un bloque "***Shift***" que pone el punto binario en su sitio original.
- El bloque "***FFT v3_1",*** está configurado según la figura (4.69).

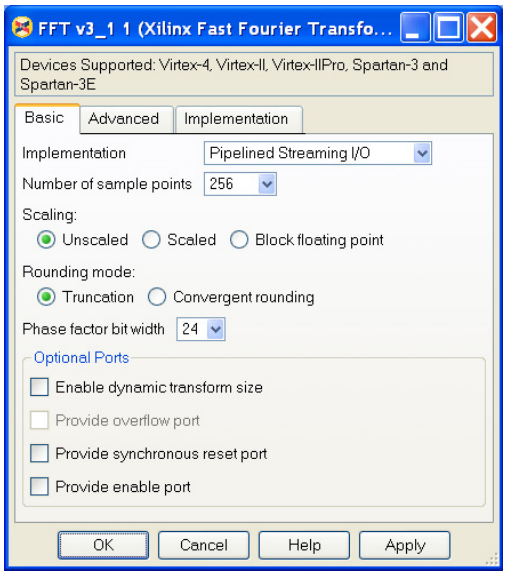
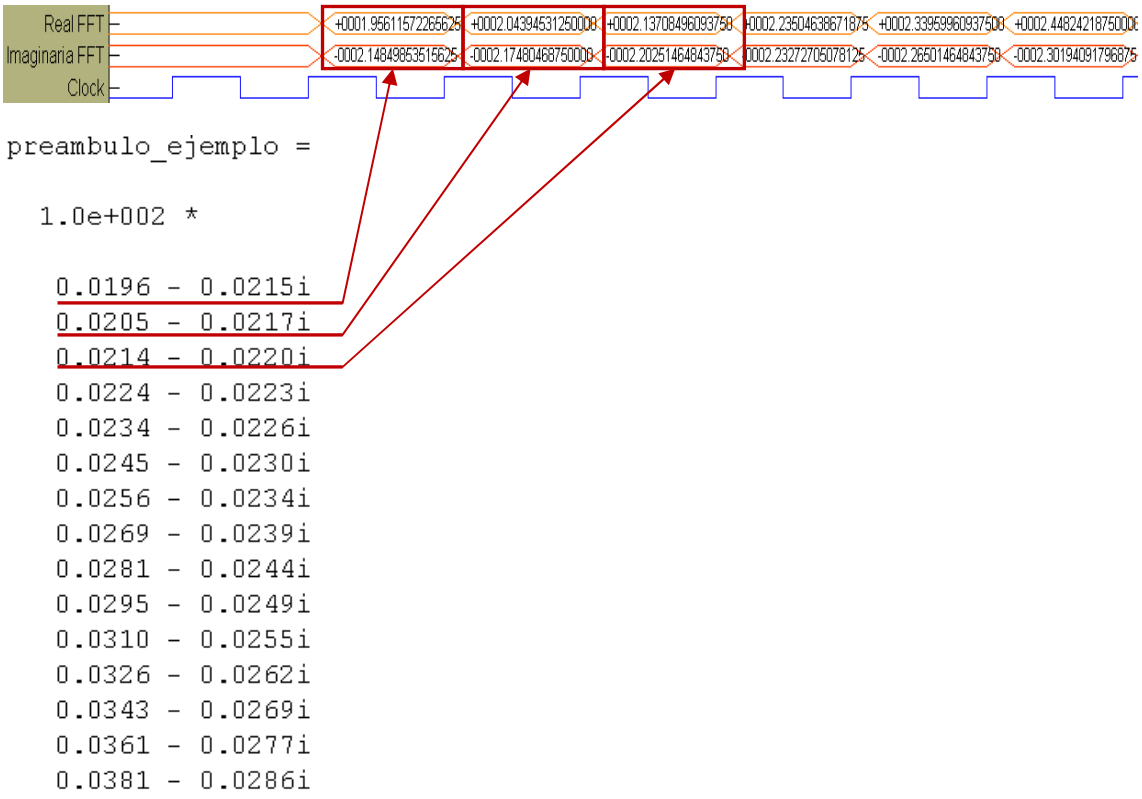


Figura 4.69: Cuadro de configuración del bloque "FFT v3_1"

- Se introduce latencia en los bloques porque el bloque empieza a leer la trama de entrada 4 ciclos de reloj después del flanco de subida de la señal en "*start*" (ver especificaciones de tiempo del bloque [4]).
- Los resultados de la simulación se ilustran en la figura (4.70), donde se puede apreciar que los resultados de la FFT coinciden con esa misma FFT realizada en Matlab. Por tanto, el bloque presenta el comportamiento esperado.




```
preambulo_ejemplo =

  1.0e+002 *

   0.0196 - 0.0215i
   0.0205 - 0.0217i
   0.0214 - 0.0220i
   0.0224 - 0.0223i
   0.0234 - 0.0226i
   0.0245 - 0.0230i
   0.0256 - 0.0234i
   0.0269 - 0.0239i
   0.0281 - 0.0244i
   0.0295 - 0.0249i
   0.0310 - 0.0255i
   0.0326 - 0.0262i
   0.0343 - 0.0269i
   0.0361 - 0.0277i
   0.0381 - 0.0286i
```

Figura 4.70: Resultados obtenidos para la salida de la FFT en Simulink® y la FFT realizada en Matlab® (Sólo se muestran las primeras muestras)

3.3.2Bloques de la librería Xilinx Reference Blockset

3.3.2.1 Bloque Xilinx CORDIC ATAN

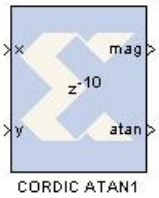


Figura 4.71: Bloque Xilinx CORDIC ATAN

Este bloque (figura (4.71)) modela un circuito que calcula el argumento (*atan*) y el módulo (*mag*) de un número complejo con parte real "x" y parte imaginaria "y". Para ello, utiliza el algoritmo iterativo CORDIC (*COrdinate Rotation DIgital Computer*) en su modo vectorización circular. Simplemente, comentar que este algoritmo está basado en la rotación de vectores. Si el lector desea más información sobre cómo funciona este algoritmo, se invita a que lea la referencia [25]. En el presente proyecto, solo nos interesará el argumento del número complejo.

- *Puertos del bloque*
  1) Puertos de entrada (*x, y*): determinan la parte real y la parte imaginaria del número complejo.
  2) Puertos de salida (*mag, atan*): determinan el módulo y el argumento del número complejo leído los puertos de entrada.

- *Configuración básica*
  Si hacemos doble clic sobre el icono de la figura (4.71), obtenemos el cuadro de configuración del bloque (figura (4.72)).

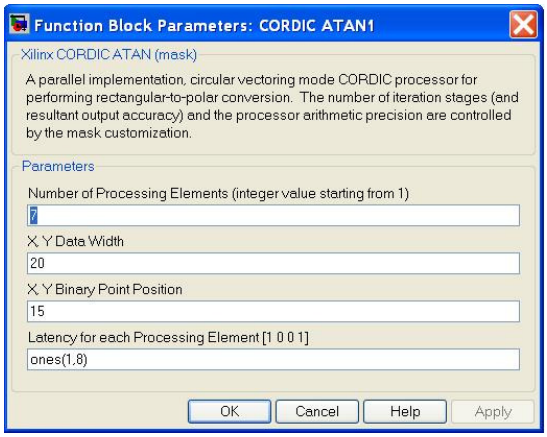


Figura 4.72: Cuadro de configuración del Bloque Xilinx CORDIC ATAN

En el cuadro de configuración mostrado en la figura (4.72), destacamos los campos: *Number of Processing Elements, (X, Y) Data Width, (X, Y) Binary Poiny Position y Latency for each processing element. Number of Processing Elements* determina el número de etapas iterativas usadas, *(X, Y) Data Width* determina el número de bits de entrada, *(X, Y) Binary Poiny Position* determina la localización del punto binario y *Latency for each processing element* determina la latencia después de cada iteración. Notar que los datos de entrada deberán ser de tipo "*con signo*" (*signed*) y deberán tener el mismo número de bits y localización de punto binario.

- *Ejemplo de utilización*

  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.73)):

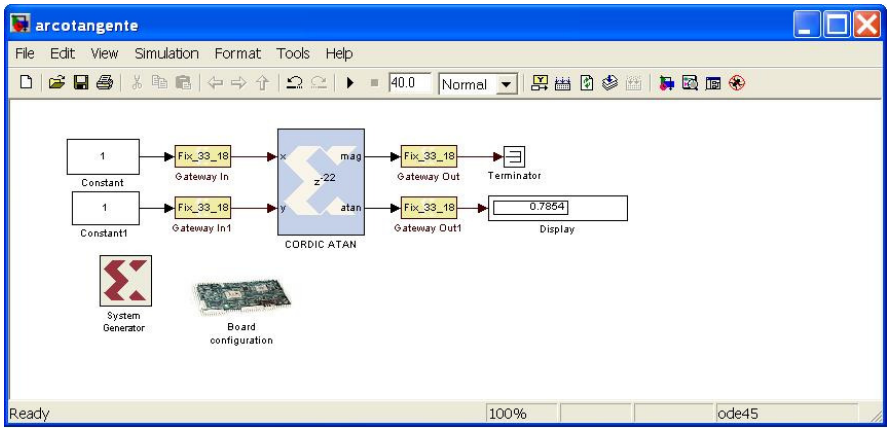


Figura 4.73: Ejemplo de utilización del Bloque Xilinx CORDIC ATAN

En este modelo se pretende mostrar el funcionamiento del bloque "***CORDIC ATAN***". En él, se calcula el argumento del número complejo $1 + 1j$. Para generar la parte real y la parte imaginaria se utilizan dos bloques "***Constant***" de la librería Simulink®, cuyas salidas son transformados a datos tipo "*con signo*" (*signed*) de 33 bits y punto fijo localizado en el bit 18 a través del los bloques "***Gateway In***". El bloque "***CORDIC ATAN***", está configurado según la figura (4.74).

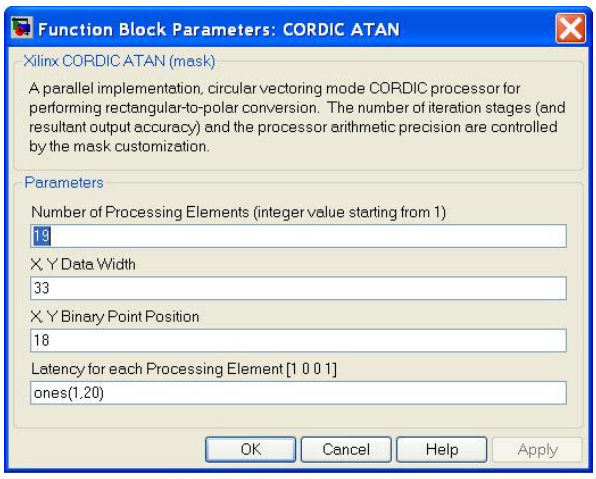


Figura 4.74: Cuadro de configuración del Bloque Xilinx CORDIC ATAN para el modelo de ejemplo

Como podemos observar de la figura (4.73), el resultado mostrado en el display es correcto, ya que el argumento del número complejo $1 + 1j$ es $\pi/4 = 0.7854$. Cabe destacar que, cuantas más iteraciones le indiquemos en el cuadro de configuración (figura (4.74)) obtendremos un argumento más preciso, sin embargo, a partir de un cierto número de iteraciones no se mejora la precisión del argumento calculado.
Notar que en el modelo también aparece el bloque "***System Generator***", el cual siempre debe aparecer en los modelos en los que se usen bloques de System Generator®.

3.3.2.2 Bloque Xilinx CORDIC SINCOS

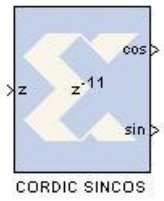


Figura 4.75: Bloque Xilinx CORDIC SINCOS

Este bloque (figura (4.75)) modela un circuito que calcula el seno y el coseno del argumento de entrada "*z*". Para ello, utiliza el algoritmo iterativo CORDIC (*COrdinate Rotation DIgital Computer*) en su modo de rotación circular. Simplemente, comentar que este algoritmo está basado en la rotación de vectores. Si el lector desea más información sobre cómo funciona este algoritmo, se invita a que lea la referencia [25].

- *Puertos del bloque*
  1) Puerto de entrada (*z*): valor del ángulo.
  2) Puertos de salida (*sin(z), cos(z)*): determinan el seno y el coseno del angula que se lee en la entrada.

- *Configuración básica*
  Si hacemos doble clic sobre el icono de la figura (4.75), obtenemos el cuadro de configuración del bloque (figura (4.76)).

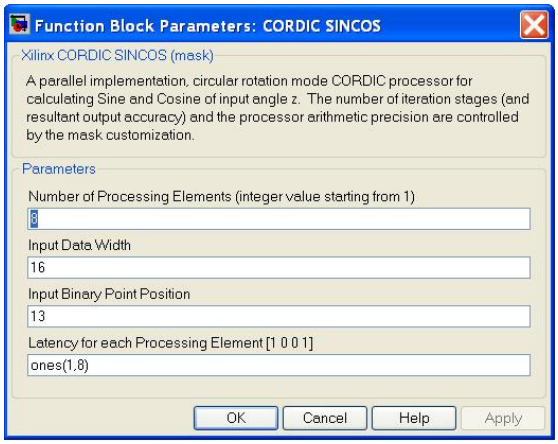


Figura 4.76: Cuadro de configuración del Bloque Xilinx CORDIC SINCOS

En el cuadro de configuración mostrado en la figura (4.76), destacamos los campos: *Number of Processing Elements, Input Data Width, Input Binary Poiny Position y Latency for each processing element. Number of Processing Elements* determina el número de etapas iterativas usadas, *Input Data Width* determina el número de bits de entrada, *Input Binary Poiny Position* determina la localización del punto binario y *Latency for each processing element* determina la latencia después de cada iteración. Notar que los datos de entrada deberán ser de tipo "con signo" (*signed*) y deberán tener el mismo número de bits y localización de punto binario.

- *Ejemplo de utilización*

  Para ilustrar como funciona este bloque, creamos el siguiente modelo en Simulink® (figura (4.77)):

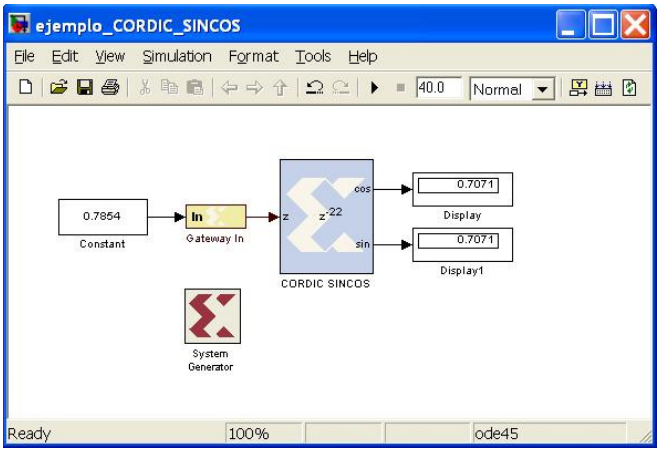


Figura 4.77: Ejemplo de utilización del Bloque Xilinx CORDIC SINCOS

En este modelo se pretende mostrar el funcionamiento del bloque "***CORDIC SINCOS***". En él, se calcula el seno y el coseno de $\pi/4 = 0.7854$. Para generar el argumento $\pi/4 = 0.7854$ se utiliza un bloque "***Constant***" de la librería Simulink®, cuya salida se transforma a un dato tipo "*con signo*" (*signed*) de 33 bits y punto fijo localizado en el bit 18 a través del bloque "***Gateway In***". El bloque "***CORDIC SINCOS***", está configurado según la figura (4.78).

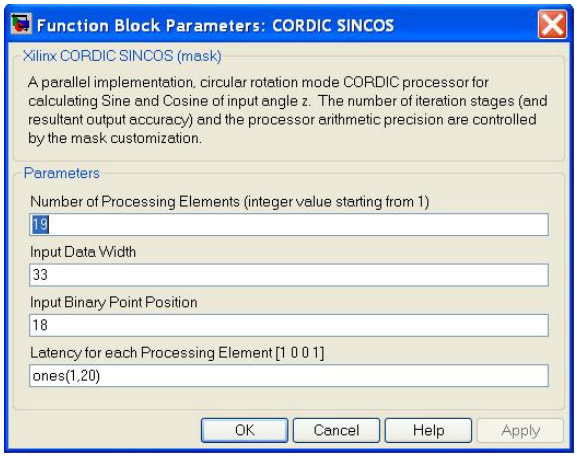


Figura 4.78: Cuadro de configuración del Bloque Xilinx CORDIC ATAN para el modelo de ejemplo

Como podemos observar de la figura (4.77), el resultado mostrado en los displays es correcto, ya que el seno y el coseno de 45º, es decir, de $\pi/4 = 0.7854$, es $1/\sqrt{2} = 07071$. Cabe destacar que, cuantas más

iteraciones le indiquemos en el cuadro de configuración (figura (4.78)) obtendremos unos resultados con mayor precisión, sin embargo, a partir de un cierto número de iteraciones no se mejora la precisión de los valores de salida.

Notar que en el modelo también aparece el bloque "***System Generator***", el cual siempre debe aparecer en los modelos en los que se usen bloques de System Generator®.

Capítulo 5

Estimación de canal (Algortimo ML-TF)

En el presente capítulo, trataremos de describir el algoritmo de estimación de canal que describiremos en hardware, a través de la utilización de las herramientas anteriormente citadas.

Para situar al lector en el ámbito que trataremos, en un primer lugar, estudiaremos, de una manera muy breve, en qué se basa la estimación de canal, para posteriormente, hacer una recopilación de distintos algoritmos de estimación de canal que podemos encontrar en la literatura. Así, se pretende dar una visión del estado del arte, en lo que a estimación de canal se refiere.

En un segundo lugar, nos centraremos en describir de manera, tanto teórica como práctica, el algoritmo concreto que utilizaremos en la estimación de canal. Se deberá comprobar el correcto funcionamiento del algoritmo de estimación de canal descrito en hardware. Así, una vez desarrollado el hardware relativo a la estimación de canal, se validará tal hardware, en distintas zonas del diseño, para comprobar que funciona correctamente

1. Estado del Arte

En todo tipo de sistema de telecomunicación inalámbrico, es preciso realizar una estimación del canal radio para poder compensar, posteriormente, los efectos perjudiciales que éste tiene sobre la señal transmitida. Normalmente, la estimación de canal, es un proceso que se realiza separadamente de otros elementos de la cadena de transmisión (igualación, decodificación detección, etc.) de un sistema de telecomunicaciones. Éste es el caso del módulo de estimación de canal que se pretende diseñar en el presente proyecto.

Podemos dividir el proceso de estimación de canal en dos partes. La primera de ellas, consistirá en la captación de información del canal, la cual se puede obtener de distintas maneras, como veremos un poco más adelante. A continuación, en una segunda parte, se utilizará esa información extraída del canal, en los distintos algoritmos que hay, para realizar la estimación del canal. Una vez que se ha estimado el canal mediante alguno de los algoritmos de

estimación existentes, es posible compensar los efectos del canal sobre la señal recibida mediante un proceso conocido como igualación. Todos estos conceptos que acabamos de comentar aparecen ilustrados en la figura (5.1).

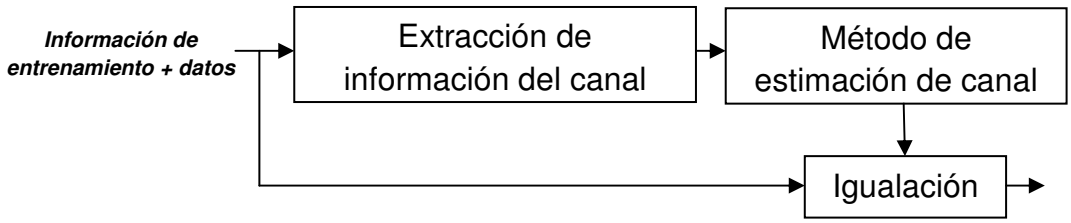


Figura 5.1: Proceso de estimación de canal

En referencia al bloque "*extracción de información del canal*" de la figura (5.1), cabe destacar que esta parte del proceso de estimación de canal se puede desempeñar de tres maneras distintas. Por un lado, existen esquemas que emplean, en transmisión, cierta información de entrenamiento (pilotos, preámbulos) conocida por el receptor (sistemas "*Data Aided*" (DA) o *Pilot Symbol Aided* (PSA)) [34], la cual permitirá saber cómo ha afectado el canal, a esa información transmitida. Éste es el caso del presente proyecto, en el cual utilizaremos un preámbulo al comienzo de la trama de datos para estimar el canal. Por otra parte, aparecen otros esquemas que no emplean información de entrenamiento (pilotos, preámbulos). Estos sistemas, conocidos como sistemas "*Decision Directed*" (DD), son esquemas dirigidos por decisión, los cuales no serán objeto del presente proyecto. Por último, existen esquemas que combinan los dos anteriores, es decir, esquemas Híbridos, los cuales tampoco consideraremos en el presente proyecto.

A partir de la información de entrenamiento (pilotos, preámbulos), se realiza la estimación de canal. Así, en el presente proyecto emplearemos, para estimar el canal, un método de máxima verosimilitud en el dominio de la frecuencia, el cual explicaremos con más detalle en los próximos apartados. Sin embargo, existe una amplia gama de métodos que sirven para realizar la estimación de canal. Por ello, nos dispondremos a citar algunos de los métodos de estimación de canal existentes en la literatura para dar una visión global del ámbito que estamos tratando.

Ye Li en [6] utiliza un estimador de canal en el dominio del tiempo basado en la inversión de una matriz cuyos elementos son la Transformada de Fourier de la correlación cruzada de los preámbulos transmitidos. La estimación de Ye Li,

trata de minimizar el error cuadrático medio (MSE, "*Mean Squared Error*"). Podemos encontrar el algoritmo empleado por Li, bajo las siglas en ingles LMMSE (*Linear Minimum-Mean Squared Error*).

En [7] se desarrolla un algoritmo de estimación de canal LS (*Least Squares*). Se trata de un estimador canal en el dominio del tiempo que utiliza un único símbolo OFDM para realizar la estimación. El artículo muestra una estimación de canal basada en el método de mínimos cuadrados LS para sistemas MIMO-OFDM.

En [8] se establece una comparación entre dos estimadores. Concretamente, compara un estimador de canal LS con uno LMMSE. En el artículo, se proponen dos tipos de preámbulo, uno normal, en el que se transmiten varios símbolos de entrenamiento, y otro cíclico (grupos de antenas). Cuando emplea el preámbulo cíclico, no se estima en todas las frecuencias el canal, por lo que propone tres métodos de interpolación para estimar el canal en las frecuencias que faltan. Estos métodos de interpolación son: interpolación lineal, interpolación LMMSE e interpolación LS basado en DFT.

En [9] se propone un estimador ML (*Maximun Likelihood*) empleando un preámbulo constituido por un único símbolo OFDM y agrupando las antenas trasmisoras en grupos que emplearán distintas subportadoras. El artículo describe un preámbulo que minimiza la MSE.

En [10] se establece una comparación entre un estimador ML en el dominio del tiempo y un estimador LS en el dominio frecuencial, utilizando un preámbulo basado en un único símbolo OFDM. Para el estimador LS se necesita realizar una interpolación lineal, ya que no se estima el canal en todas las frecuencias.

En [11] se comparan tres estimadores empleados en la estimación de canales con varios "*taps*" o etapas aproximadamente cero. Simplemente, enumeraremos los métodos comparados: LS, *Constraint Least Square Channel Estimation* (CLSCE) y *Matching Pursuit Channel Estimation* (MPCE).

En [12] se estudia un estimador LMMSE. Para la realizar la estimación de canal, las distintas antenas transmisoras emiten una secuencia de entrenamiento basada en el desplazamiento cíclico de una secuencia inicial. El artículo discute el coste computacional de usar este estimador en el dominio frecuencial o temporal.

El artículo [13] trata de minimizar el error cuadrático medio, mediante el empleo de un filtro de Wiener. Se estudia también distintos esquemas de entrenamiento (información en todas las subportadoras o solo en algunas).

En [14] se desarrolla un estimador de canal LS que evita tener que realizar la operación de inversión de matrices. El artículo describe una distribución de tonos piloto óptima, la cual minimiza el error cuadrático medio de la estimación. Esta distribución de pilotos es espacialmente utilizada en canales que varían rápidamente.

En [15] se desarrolla un estimador muy robusto ante fuertes interferencias (*MultiTone NarrowBand Interference*, MTNBI).

2. Algoritmo empleado en la estimación de canal (Explicación Teórica)

En el presente apartado, se explicará teóricamente el algoritmo empleado en la estimación de canal, el cual será descrito en hardware posteriormente. Concretamente, se ha elegido el algoritmo de máxima verosimilitud "*Maximun Likelihood Time-Frequency*" (ML-TF) presentado en la referencia bibliográfica [2].

El algoritmo ML-TF, trata de estimar el canal utilizando las dimensiones de tiempo y frecuencia de los símbolos OFDM transmitidos como preámbulo. Los símbolos OFDM que componen el preámbulo, también conocidos como secuencia de entrenamiento, serán transmitidos al comienzo de la transmisión como cabecera de la trama de datos. Así, para explicar de una forma genérica como funciona este algoritmo, supondremos que esta información de entrenamiento consistirá en L_P símbolos OFDM conocidos.

Si adaptamos la relación entrada-salida del canal, la cual deducimos en la expresión (2.16) (Capítulo 2), para el l-ésimo símbolo OFDM del preámbulo o secuencia de entrenamiento ($l = 0, \dots, L_P - 1$), se tiene la siguiente expresión (solo referida al preámbulo):

$$\underline{R}^l(k) = \underline{\underline{H}}(k)\underline{P}^l(k) + \underline{W}^l(k) \qquad (5.1)$$

donde $\underline{P}^l(k)$ es un vector que recoge los símbolos complejos del preámbulo emitido por las antenas transmisoras (índices del vector, $j = 1, \dots, M_T$) en la k-ésima subportadora para el l-ésimo símbolo OFDM del preámbulo ($\underline{P}^l(k) = \left[P_1^l(k), \dots, P_{M_T}^l(k)\right]^T$), $\underline{\underline{H}}(k)$ es la matriz de canal para la k-ésima subportadora

de dimensión $M_T\ x\ M_R$, donde se tienen los coeficientes de canal en el dominio de la frecuencia de los $M_T M_R$ canales que componen el canal MIMO en la subportadora "k", y $\underline{R^l}(k)$ es un vector que recoge el preámbulo recibido en cada antena receptora (índices del vector, i= $1, \dots, M_T$) en la k-ésima subportadora para el l-ésimo símbolo OFDM del preámbulo ($\underline{R^l}(k) = \left[R_1^l(k), \dots, R_{M_R}^l(k)\right]^T$). $\underline{W^l}(k)$ hace referencia a las componentes de ruido gaussiano.

La expresión (5.1), solo considera un símbolo OFDM del preámbulo, por que será necesario considerar todos los símbolos que constituyen el preámbulo. Así, considerando los L_P símbolos OFDM del preámbulo completo, el preámbulo transmitido se puede definir mediante una matriz $\underline{\underline{P}}$ de dimensión $M_T\ x\ L_P$:

$$\underline{\underline{P}}(k) = \left[\underline{P^0}(k), \dots, \underline{P^{L_P-1}}(k)\right] \qquad (5.2)$$

Así mismo, el preámbulo recibido en cada antena receptora se podrá definir mediante una matriz $\underline{\underline{R}}$ de dimensión $M_R\ x\ L_P$, de manera que podemos extender la expresión (5.1) en la siguiente expresión que relaciona la entrada y la salida del canal de comunicación en el dominio de la frecuencia:

$$\underline{\underline{R}}(k) = \underline{\underline{H}}(k)\underline{\underline{P}}(k) + \underline{W^l}(k) \qquad (5.3)$$

Para estimar los $M_T M_R$ canales representados en la matriz de canal $\underline{\underline{H}}(k)$, es necesario, como mínimo, realizar $M_T M_R$ medidas, ya que en cada símbolo OFDM del preámbulo se incluyen M_R medidas. Así, se debe cumplir la siguiente relación para una buena estimación del canal [2]:

$$L_P \geq M_T \qquad (5.4)$$

La estimación de canal tiene como objetivo estimar la matriz $\underline{\underline{H}}(k)$ de la manera más precisa posible. Para ello, el algoritmo de estimación de canal que estamos tratando se basa en un criterio de máxima verosimilitud ("*Maximun Likelihood*", ML) en el dominio de la frecuencia. Bajo condiciones de ruido gaussiano, se puede estimar la matriz de canal $\underline{\underline{H}}(k)$, a partir del preámbulo recibido, mediante la minimización de la norma cuadrática respecto de $\underline{\underline{H}}(k)$ [2]:

$$\underline{\underline{\widehat{H}}}_{ML}(k) = argmin_{\underline{H}(k)} \left\| \underline{\underline{R}}(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k) \right\|_F^2 \qquad (5.5)$$

donde $\underline{\underline{\widehat{H}}}_{ML}(k)$ es la matriz de canal estimada para la k-ésima subportadora y los dos corchetes dobles hacen referencia a la *norma de Frobenius* de la matriz que hay en su interior ($\|X\|$ es la norma de Frobenius de la matriz X).

De la expresión (5.5), se deriva la siguiente función de costes:

$$C^2 = \left\| \underline{\underline{R}}(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k) \right\|_F^2 = tr\left[\underline{\underline{R}}(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k)\right]\left[\underline{\underline{R}}(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k)\right]^H \qquad (5.6)$$

donde tr indica la traza de la matriz que hay entre los corchetes.

A continuación, derivamos la función de coste de (5.6) e igualamos a cero la derivada para obtener qué matriz de canal $\underline{\underline{H}}(k)$ hace mínima la función de coste:

La derivada de (5.6) es:

$$\delta(C^2) = tr\left(-\delta\underline{\underline{H}}(k)\underline{\underline{P}}(k)\left[\underline{\underline{R}}(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k)\right]^H - \left[\underline{\underline{R}}(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k)\right]\underline{\underline{P}}^H(k)\delta\underline{\underline{H}}^H(k)\right)$$
$$= tr\left(\delta\underline{\underline{H}}(k)\left[\underline{\underline{P}}(k)\underline{\underline{R}}^H(k) - \underline{\underline{P}}(k)\underline{\underline{P}}^H(k)\underline{\underline{H}}^H(k)\right] - \left[\underline{\underline{R}}(k)\underline{\underline{P}}^H(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k)\underline{\underline{P}}^H(k)\right]\delta\underline{\underline{H}}^H(k)\right) \qquad (5.7)$$

Si igualamos la expresión (5.7) a cero:

$$\underline{\underline{R}}(k)\underline{\underline{P}}^H(k) - \underline{\underline{H}}(k)\underline{\underline{P}}(k)\underline{\underline{P}}^H(k) = 0 \qquad (5.8)$$

de manera que se obtiene:

$$\underline{\underline{H}}(k) = \underline{\underline{R}}(k)\underline{\underline{P}}^H(k)\left[\underline{\underline{P}}(k)\underline{\underline{P}}^H(k)\right]^{-1} \qquad (5.9)$$

Siempre y cuando se pueda calcular la matriz inversa de la expresión (5.9), el estimador de máxima verosimilitud del canal es:

$$\underline{\underline{\widehat{H}}}_{ML}(k) = \underline{\underline{R}}(k)\underline{\underline{P}}^H(k)\left[\underline{\underline{P}}(k)\underline{\underline{P}}^H(k)\right]^{-1} \qquad (5.10)$$

En la práctica, se puede definir una matriz $T(k)$, la cual se calculará previamente mediante el preámbulo conocido. Esta matriz para la k-ésima

subportadora, se extrae de la expresión (5.10) y se define de la siguiente manera:

$$\underline{\underline{T}}(k) = \underline{\underline{P^H}}(k)\left[\underline{\underline{P}}(k)\underline{\underline{P^H}}(k)\right]^{-1} \qquad (5.11)$$

Así, el estimador de máxima verosimilitud queda:

$$\underline{\underline{\widehat{H}}}_{ML}(k) = \underline{\underline{R}}(k)\underline{\underline{T}}(k) \qquad (5.12)$$

Como podemos deducir de la expresión (5.12), la estimación de canal para cada subportadora se hace mediante una simple multiplicación de matrices, es decir, mediante la multiplicación del preámbulo recibido $\underline{\underline{R}}(k)$ y la matriz $T(k)$, la cual puede ser calculada y almacenada previamente, pues solo utiliza información conocida por el transmisor y receptor que no varia en el tiempo. El hecho de que $T(k)$ pueda calcularse previamente y almacenarse, en la práctica, demuestra la sencillez de este algoritmo de estimación de canal, reduciéndose el coste hardware que lleva asociado el cálculo de la matriz inversa de la expresión (5.10).

3. Diseño del módulo de estimación de canal

El diseño del módulo que desempeña el algoritmo de estimación de canal, ha sido realizado utilizando bloques de la librería *Xilinx Blockset de Xilinx System Generator for DSP®* [4], los cuales han sido descritos en el capítulo 4. El algoritmo que se ha descrito en hardware se corresponde con el algoritmo ML-TF anteriormente explicado. Este módulo de estimación de canal está dividido en tres bloques principales, los cuales se muestran en la figura (5.2).

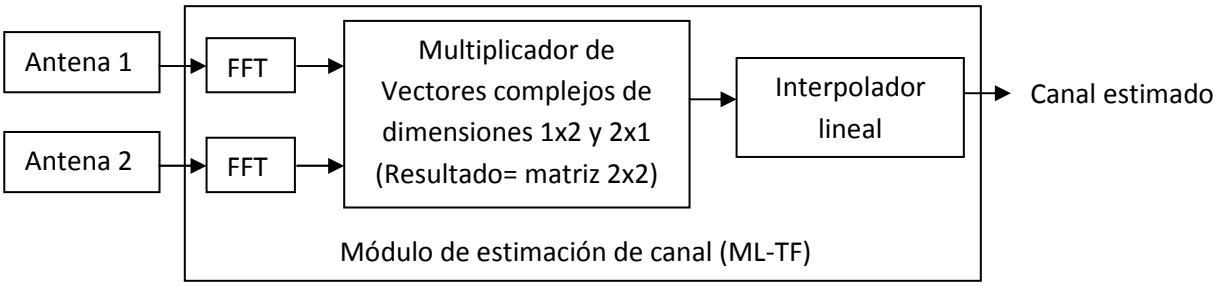


Figura 5.2: Visión general del módulo de estimación de canal

Antes de comenzar con la descripción del módulo de estimación de canal, es preciso tener en cuenta, previamente, una serie de consideraciones generales:

- Para comprobar el correcto funcionamiento del hardware diseñado, es preciso generar las señales de entrada a dicho hardware. Las señales de entrada deberán de corresponderse con el preámbulo recibido, una vez éste ha atravesado el canal. Para ello, se ha simulado en Matlab® el paso del preámbulo a través del canal. La función de Matlab® que simula el paso del preámbulo por el canal es "*MLTF_offset.m*", la cual realiza una llamada a otra función de Matlab® ("*chan_SUI.m*") que simula uno de los 4 canales SISO del canal 2x2 MIMO. Esta función, nos devolverán la parte real y la parte imaginaria de los preámbulos recibidos en cada antena receptora en los vectores "*En_re_Pre_out_ant1_time, En_im_Pre_out_ant1_time, En_re_Pre_out_ant2_time y En_im_Pre_out_ant2_time*". Seremos capaces de importar los datos guardados en esos vectores generados por la función de Matlab® mediante el uso de bloques "***From Workspace***".

- Las dos funciones de Matlab®, anteriormente comentadas, las podemos encontrar en el Anexo A.

- El canal simulado en la función de Matlab® "*chan_SUI.m*" se corresponde con un modelo de canal "*Standford University Interim*" (SUI) modificado. Concretamente, se trata del modelo de canal SUI-3 modificado. En la referencia bibliográfica [16], se detallan las características de este modelo de canal, por lo que se invita al lector a consultar tal documento, si desea conocer más acerca de este modelo de canal.

- Para definir los puertos de entrada al hardware diseñado, se han utilizado bloques "***Gateway In***" configurados para trabajar con datos tipo "*signed*" de 24 bits con el punto binario en el bit 14. Esto se ha realizado así, para poder representar de una manera precisa la información, de lo contrario la calidad y precisión de la estimación de canal no sería la misma. Este número de bits se trata de mantener a lo largo del diseño.

- El sincronismo de todos los elementos síncronos del módulo de estimación de canal está controlado por un contador común. Este contador común, será utilizado para generar las señales de control que gestionan distintos elementos del diseño.

➔ *Visión general del modelo Simulink®*

En la figura (5.3), mostramos el aspecto general que tiene este módulo de estimación de canal. Sobre ella, resaltaremos las partes más importantes mediante recuadros y las numeraremos para describirlas más adelante.

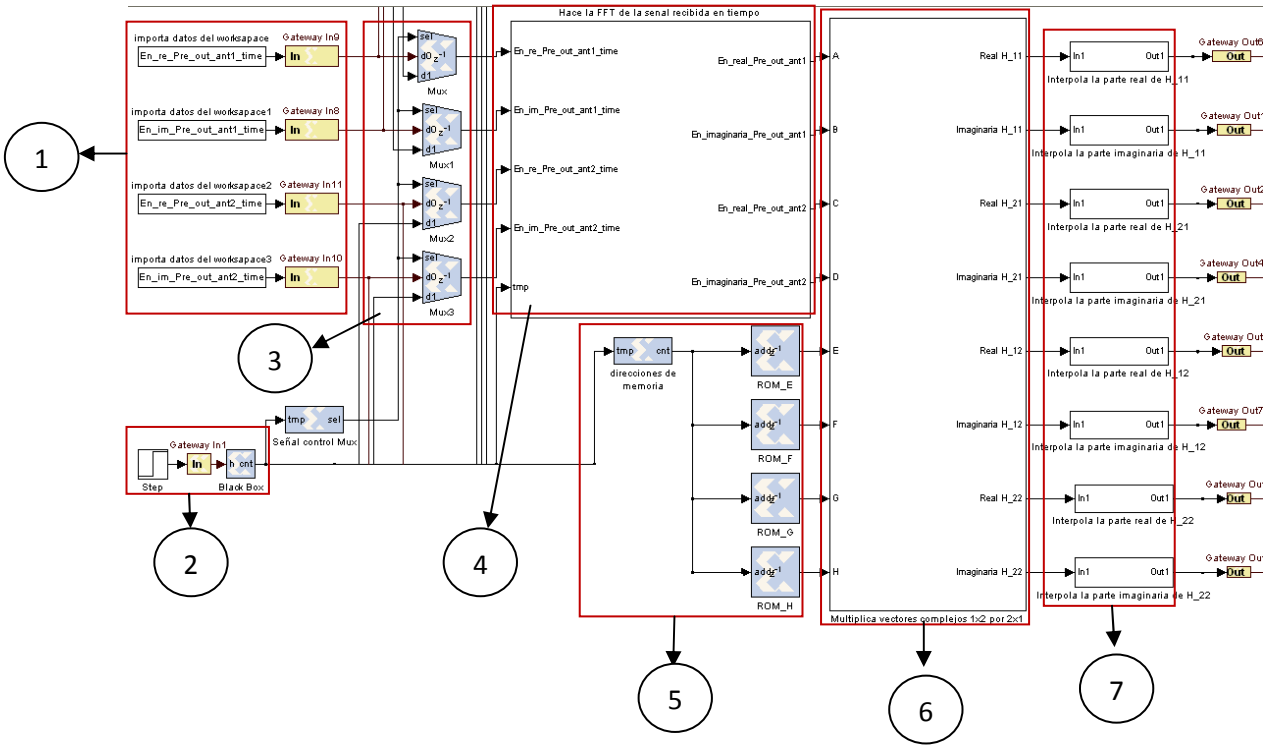


Figura 5.3: Aspecto del modelo Simulink® de estimación de canal

3.1 Entrada de datos

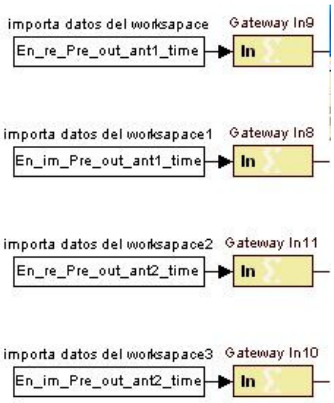


Figura 5.4: Entrada de datos

En esta etapa del diseño (figura (5.4)) se importan del "*workspace*" de Matlab® los resultados de la simulación del canal. Estas cuatro entradas hacen referencia al preámbulo recibido en cada antena receptora. Se importa la parte real del preámbulo recibido en la antena 1, la parte imaginaria del preámbulo recibido en la antena 1, la parte real del preámbulo recibido en la antena 2 y la parte imaginaria del preámbulo recibido en la antena 2.

Cada bloque "***From Workspace***" está configurado según la figura (5.5).

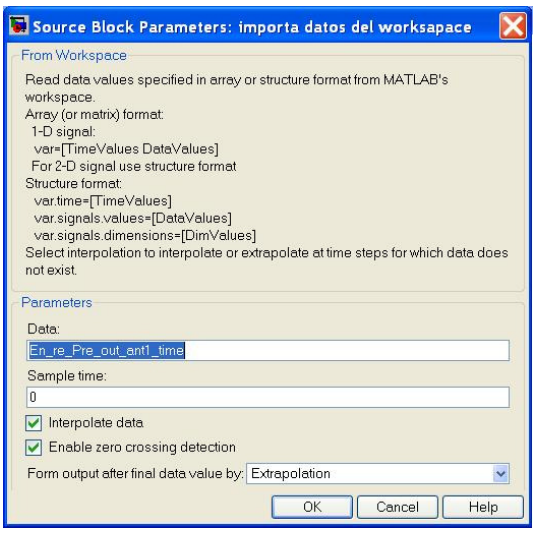


Figura 5.5: Cuadro de configuración de los bloques "From Workspace"

También, se definen las entradas a nuestro hardware mediante bloques "***Gateway In***" configurados según la figura (5.6).

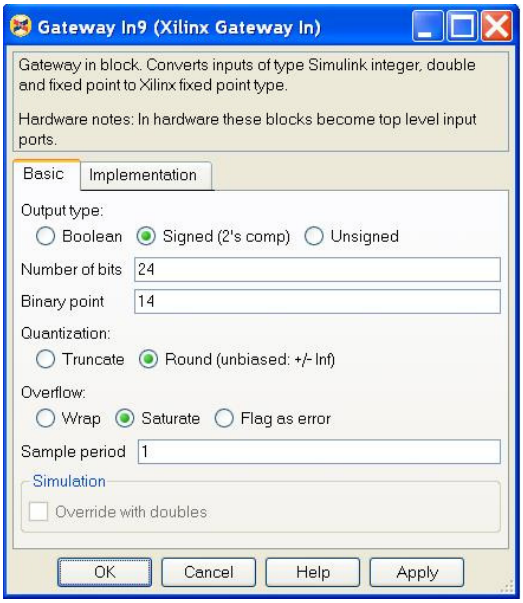


Figura 5.6: Cuadro de configuración de los bloques "Gateway In"

3.2 Contador Común

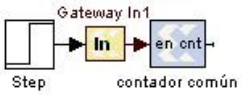


Figura 5.7: Contador Común

Este bloque (figura (5.7)) se trata de un contador de 32 bits que sirve para controlar el sincronismo de todo el diseño. El bloque posee una entrada de habilitación y una salida de 32 bits tipo "*unsigned*". Cuando la entrada de habilitación está en nivel alto, el contador comienza la cuenta y cuando está a nivel bajo el contador se pone a cero. Esta entrada de habilitación puede servir para activar y desactivar este módulo de estimación de canal, hasta que se reciba el preámbulo. La señal que indica la llegada de la trama transmitida en el módulo se sincronización temporal, puede ser empleada para activar esta señal de habilitación y comenzar la estimación de canal.

Este bloque ha sido diseñado mediante un bloque "***Black Box***", a través del cual podemos asociar código VHDL al modelo Simulink®. El código VHDL de este contador lo podemos encontrar en el anexo B.

El bloque "***Black Box***" que contiene este contador, está configurado según la figura (5.8).

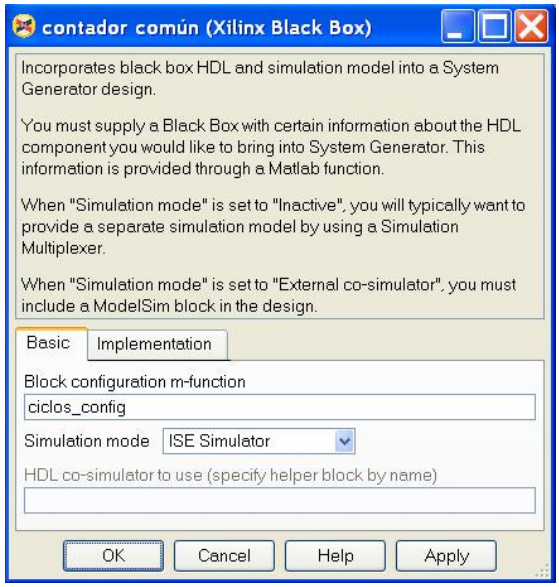


Figura 5.8: Cuadro de configuración del bloque "Black Box"

La entrada de habilitación del contador está configurada según la figura (5.9).

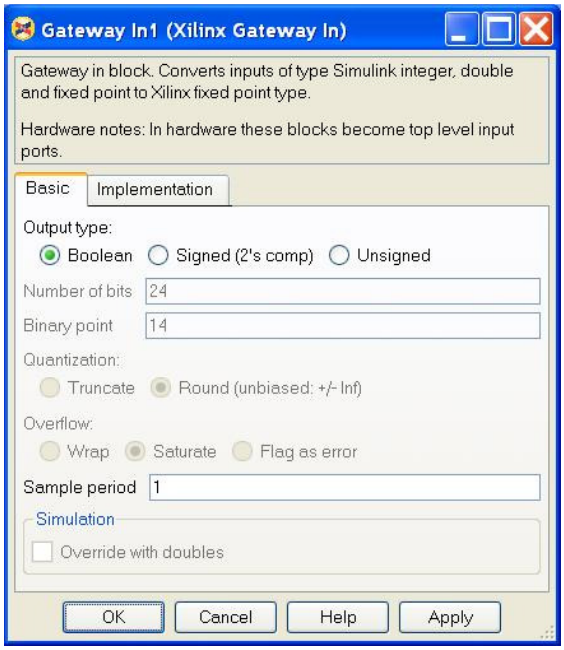


Figura 5.9: Cuadro de configuración del bloque "Gateway In" (entrada de habilitación)

3.3 Multiplexores

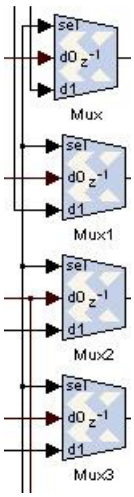


Figura 5.10: Multiplexores

Esta etapa del diseño (figura (5.10)), es empleada para volver a estimar el canal una vez corregido el "*offset*" en frecuencia en el módulo pertinente. Se requerirá un camino de realimentación para poder reestimar el canal. La función de estos multiplexores será seleccionar el camino éste realimentación. Esta etapa la veremos más en detalle en el capítulo de sincronización frecuencial, por lo que no profundizaremos más en ella.

3.4 FTT de la señal recibida en el dominio del tiempo

La función de Matlab® que simula el paso del preámbulo por el canal, nos devuelve el preámbulo recibido en el dominio del tiempo. Sin embargo, el algoritmo ML-TF, trabaja en el dominio de la frecuencia, por lo tanto será necesario obtener el preámbulo recibido en el dominio de la frecuencia.

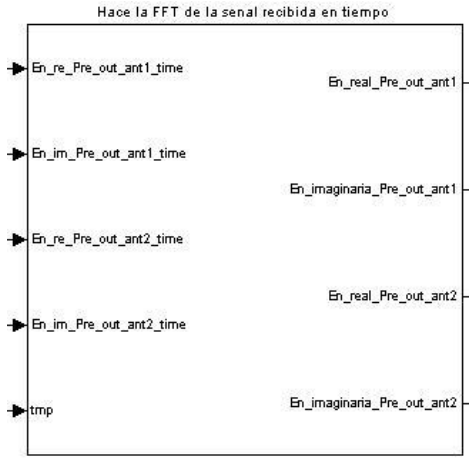


Figura 5.11: Hace la FFT de la señal recibida en el tiempo

Esta etapa del diseño (figura (5.11)), trata de obtener el preámbulo recibido en el dominio de la frecuencia mediante el empleo de dos bloques "***FFT v3_1***". Existe un bloque "***FFT v3_1***" para cada antena receptora, es decir, en esta etapa del diseño tendremos dos bloques "***FFT v3_1***".

Cada bloque "***FFT v3_1***", está configurado para realizar la FFT de 256 puntos sobre una trama de 256 muestras. Así, obtendremos el preámbulo recibido en el dominio de la frecuencia.

En la figura (5.12), se ilustra cómo están conectados los bloques "***FFT v3_1***" de esta etapa del diseño.

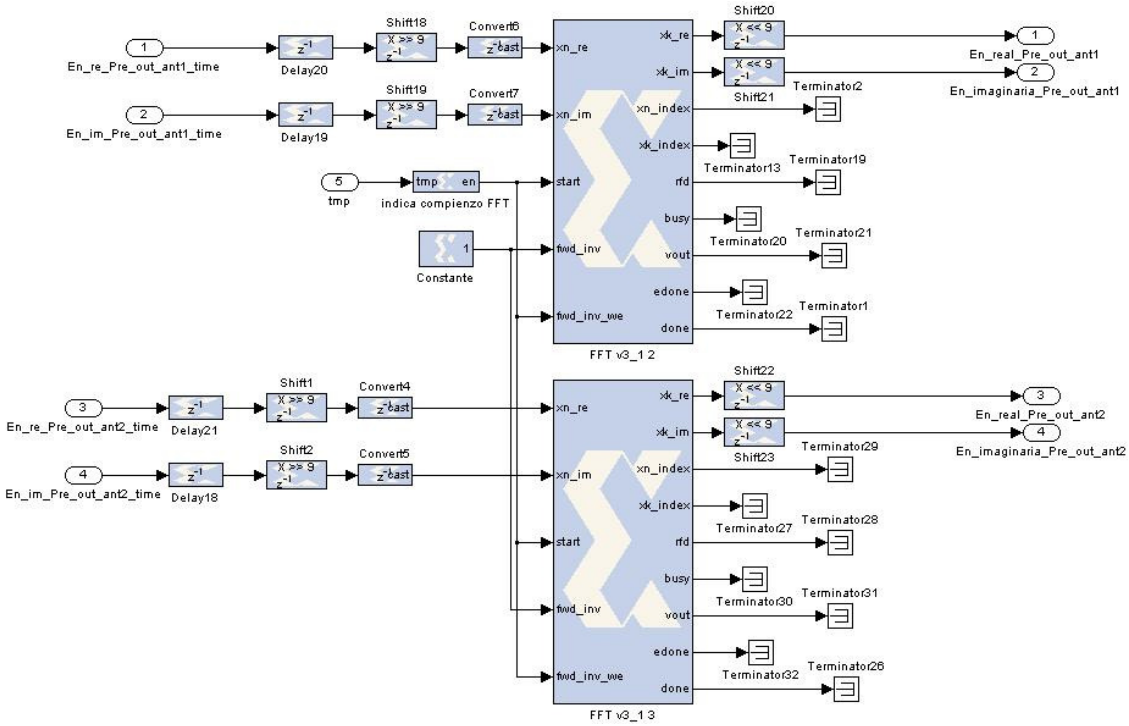


Figura 5.12: Interior del subsistema "Hace la FFT de la señal recibida en el tiempo"

Es de destacar en la figura (5.12) como están conectadas las entradas de los bloques "**FFT v3_1**". Los aspectos que debemos destacar del bloque "**FFT v3_1**" son:

➔ La entrada "*Start*" está conectada a una entrada de habilitación que determina cuando el bloque "***FFT v3_1***" debe comenzar a capturar la trama de entrada para comenzar a realizar la FFT. Esta entrada de habilitación está generada por el bloque "*indica comienzo de la FFT*". Este bloque tiene como entrada de datos la salida del contador común, anteriormente descrito, de manera que cuando el contador alcanza un determinado valor, el bloque genera la señal de habilitación que determina el comienzo de la FFT.

El bloque "*indica comienzo FFT*" es un bloque "***Black Box***", el cual tiene asociado un código VHDL que podemos encontrar en el anexo B. El cuadro de configuración de este bloque lo podemos encontrar en la figura (5.13).

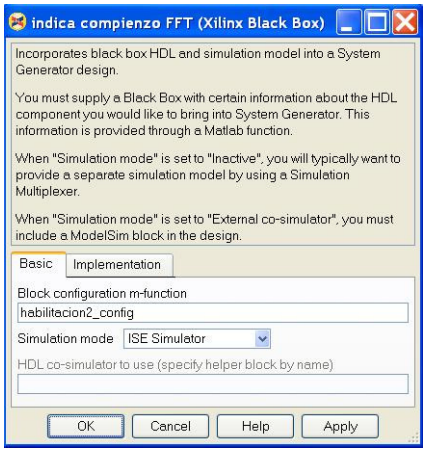


Figura 5.13: Cuadro de configuración del bloque "Black Box" ("Indica comienzo de la FFT")

➔ Cada bloque "***FFT v3_1***" tiene una entrada para la parte real y otra para la parte imaginaria (*xn_re* y *xn_im*). Los datos en estas dos entradas necesitarán ser adaptados. Para ello, hay que tener en cuenta que, el bloque "***FFT v3_1***" admite datos de entrada tipo "*signed*" de 8, 12, 16, 20 o 24 bits con el punto binario situado de tal manera que solo tenemos un bit para representar la parte entera del dato. Este efecto, se soluciona adaptando los datos de entrada de la siguiente manera:

- >> En primer lugar utilizaremos bloques "***Shift***", con los que realizaremos un desplazamiento lógico a la derecha y un aumento del número de bits.

A la salida habrá que hacer la operación inversa para tener el mismo tipo de dato que teníamos a la entrada, es decir, un dato tipo "*signed*" de 24 bits con el punto binario en el bit 14.

En las figuras (5.14) y (5.15), se ilustra cómo están configurados los bloques "***Shift***", tanto a la salida como a la entrada del bloque "***FFT v3_1***".

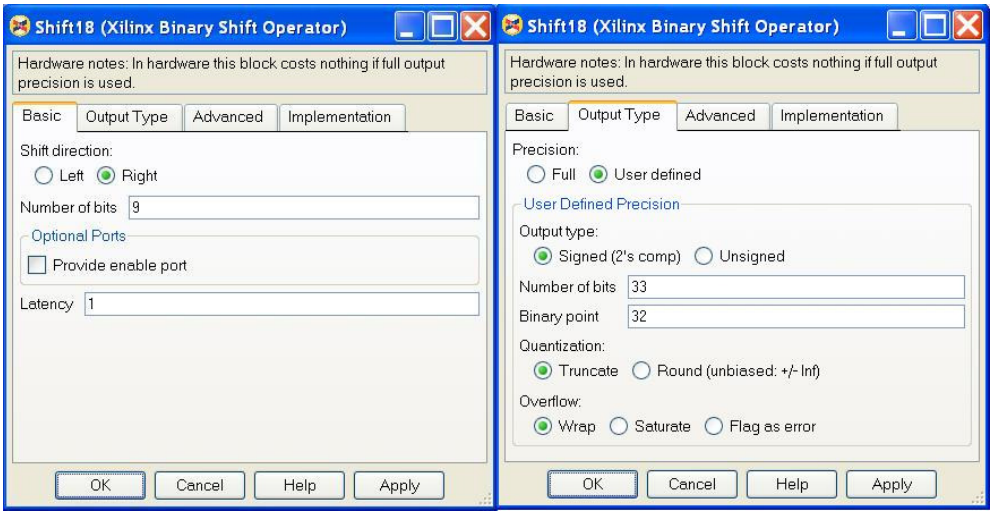


Figura 5.14: Cuadro de configuración del bloque "Shift" a la entrada del bloque FFT v3_1

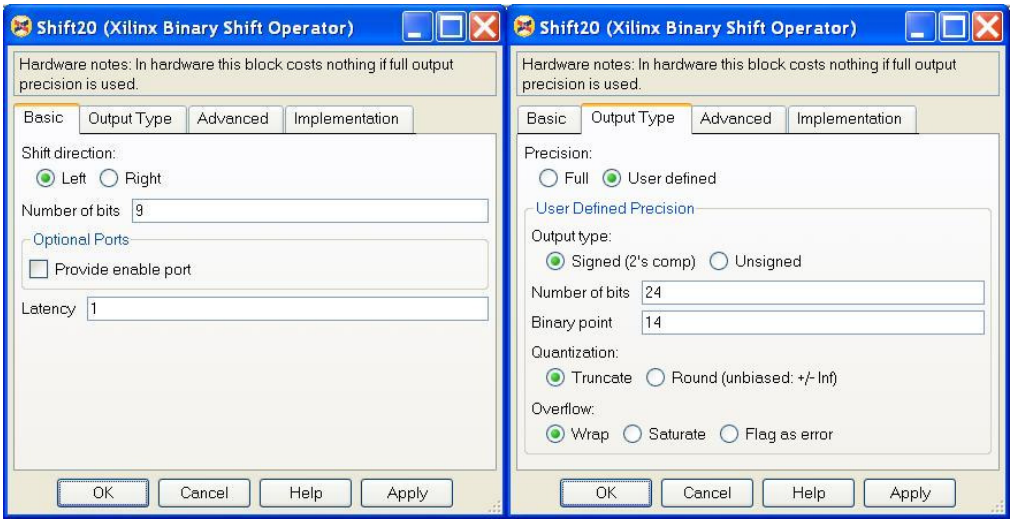


Figura 5.15: Cuadro de configuración del bloque "Shift" a la salida del bloque FFT v3_1

->> En un segundo lugar, emplearemos bloques "***Convert***" para adaptar el número de bits de los datos de entrada al bloque "***FFT v3_1***".

En la figura (5.16), se ilustra cómo están configurados los bloques "***Convert***" utilizados.

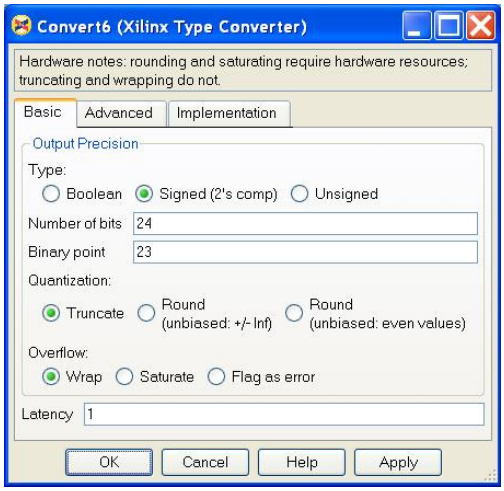


Figura 5.16: Cuadro de configuración del bloque "Convert" a la entrada del bloque FFT v3_1

➔ La entrada "*fw_inv*", indica si el bloque está configurado para realizar una FFT o una IFFT. En este caso, como se quiere hacer una FFT, utilizamos como dato de entrada una constante de valor 1 tipo booleano.

➔ La entrada "*fw_inv_we*" se utiliza para cargar el valor de la entrada "*fw_inv*", que indicará si se realiza una transformada inversa o no. Cuando esta activa, se carga el valor de "*fw_inv*". Como podemos observar en la figura (5.12), la entrada "*fw_inv_we*" está conectada al bloque "*indica comienzo FFT*" para que se cargue el tipo de transformada en el mismo instante que se habilita el bloque para que comience a realizar la FFT.

En la figura (5.17), podemos observar cómo está configurado uno de los bloques "***FFT v3_1***" de esta etapa del diseño.

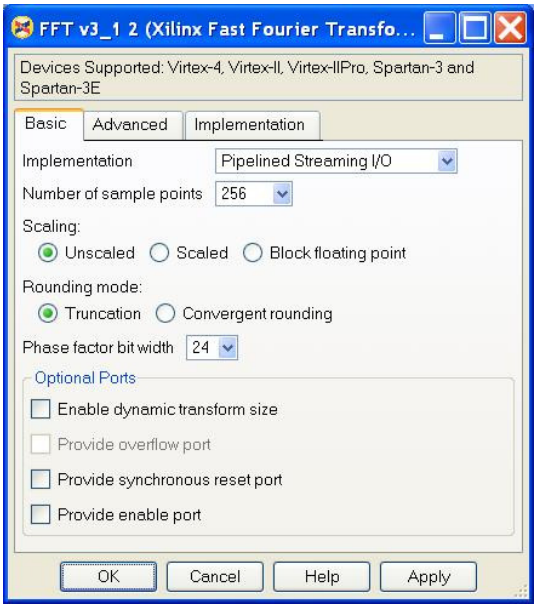


Figura 5.17: Cuadro de configuración del bloque FFT v3_1

Esta etapa nos permitirá tener a su salida el preámbulo recibido en el dominio de la frecuencia, para que pueda ser manipulado en el algoritmo ML-TF.

3.5 Almacenamiento de la Matriz $\underline{T}(k)$

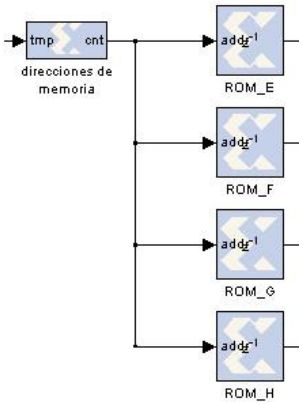


Figura 5.18: Almacenamiento de la Matriz $\underline{T}(k)$

Esta etapa del diseño (figura (5.18)), se encarga de almacenar la matriz $\underline{\underline{T}}(k)$, la cual queda definida en la expresión (5.11) (ver explicación teórica del algoritmo ML-TF). La matriz $\underline{\underline{T}}(k)$, es una matriz de dimensión 1x2, siempre y cuando, solo nos estemos refiriendo a una subportadora "k". Sin embargo, si se consideran todas las subportadoras, esta matriz se convierte en una matriz de dimensiones 256 x 2, donde cada fila está referida a una subportadora. Esta matriz que considera todas las subportadoras, la denotaremos por $\underline{\underline{T}}$. Esta matriz $\underline{\underline{T}}$ está compuesta por elementos que son números complejos, por lo que cada elemento dispondrá de una parte real y una parte imaginaria (ver Tabla (5.1)).

Posición de memoria (subportadora "k")		
0	a+b*j	l-d*j
1	w-h*j	k+t*j
…	…	…
255	g+n*j	q+x*j

Tabla 5.1: Ejemplo de distribución de los elementos de la matriz $\underline{\underline{T}}$

Partiendo de la matriz $\underline{\underline{T}}$, podemos desglosar esta matriz en otra matriz $\underline{\underline{T}}real$ de números reales de dimensión 256 x 4, donde la primera columna se corresponde con la parte real de los elementos complejos de la primera columna de $\underline{\underline{T}}$, la segunda columna se corresponde con la parte imaginaria de los elementos complejos de la primera columna de $\underline{\underline{T}}$, la tercera columna se corresponde con la parte real de los elementos complejos de la segunda columna de $\underline{\underline{T}}$ y la cuarta columna se corresponde con la parte imaginaria de los elementos complejos de la segunda columna de $\underline{\underline{T}}$. Esto que acabamos de describir, se ilustra en la Tabla (5.2).

Posición de memoria (subportadora "k")				
0	a	b	L	-d
1	w	-h	K	t
…	…	…	…	…
255	g	n	Q	x

Tabla 5.2: Ejemplo de distribución de los elementos de la matriz $\underline{\underline{T}}real$

Así pues, siguiendo la estructuración de la tabla (5.2), para almacenar la matriz $\underline{\underline{T}}real$ (indirectamente estamos almacenando $\underline{\underline{T}}$), utilizaremos 4 memorias ROM, tal y como es muestra en la figura (5.18). El almacenamiento es el siguiente:

->> En el bloque "***ROM***" llamado *ROM_E*, almacenaremos la primera columna de $\underline{\underline{T}}real$ (partes reales), la cual dispone de 256 elementos.

->> En el bloque "***ROM***" llamado *ROM_F*, almacenaremos la segunda columna de $\underline{\underline{T}}real$ (partes imaginarias), la cual dispone de 256 elementos.

->> En el bloque "***ROM***" llamado *ROM_G*, almacenaremos la tercera columna de $\underline{\underline{T}}real$ (partes reales), la cual dispone de 256 elementos.

->> En el bloque "***ROM***" llamado *ROM_H*, almacenaremos la cuarta columna de $\underline{\underline{T}}real$ (partes imaginarias), la cual dispone de 256 elementos.

En la figura (5.19), se ilustra cómo está configurado uno de los bloques "***ROM***" (el resto tiene la misma configuración, solo que almacenan datos distintos).

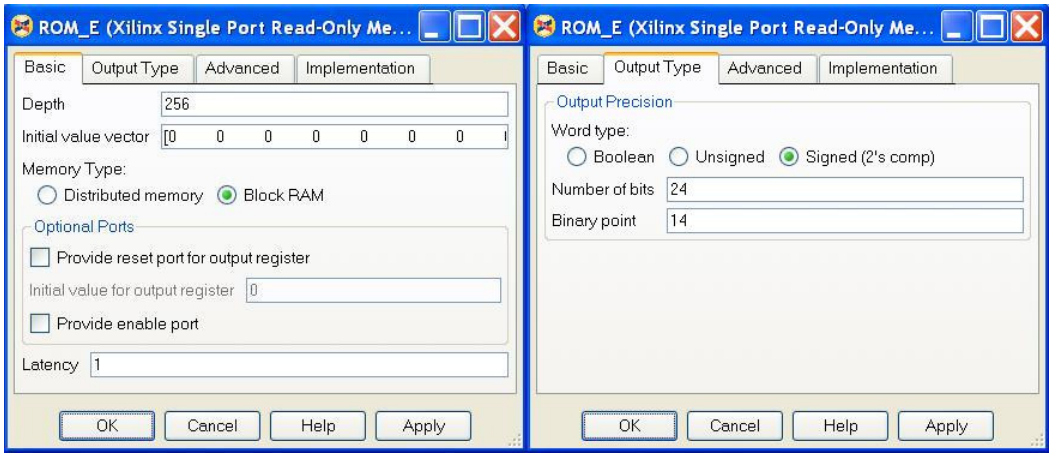


Figura 5.19: Cuadro de configuración de uno de los bloques ROM

Por otra parte, las entradas de las memorias ROM, están conectadas a un contador de 0 a 255 que es el encargado de proporcionar la dirección de memoria donde se debe leer en cada instante. El conexionado de esta etapa, permite que todas las memorias se lean en paralelo, de la dirección de memoria 0 a la 255. Esto, indirectamente, supone ir leyendo la matriz $\underline{\underline{T}}$ fila a fila, o lo que es lo mismo, subportadora a subportadora. Esta forma de leer las memorias, supone que se pueda realizar la multiplicación de vectores definida en la expresión (5.12) (ver explicación teórica del algoritmo ML-TF).

El contador que genera las direcciones de memoria está controlado por el contador común del sistema, de manera que cuando éste lleva cierto número de ciclos de reloj contados, el contador comienza la cuenta de 0 a 255. Este contador, identificado bajo el nombre "*direcciones de memoria*", esta descrita a través de un bloque "***Black Box***", el cual tiene asociado un código VHDL que podemos encontrar en el anexo B.

En la figura (5.20), se muestra como está configurado el contador que genera las direcciones de memoria (bloque "*direcciones de memoria*").

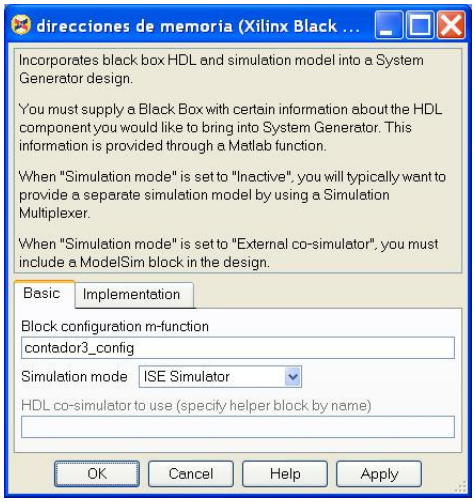


Figura 5.20: Cuadro de configuración del bloque Black Box (bloque "direcciones de memoria")

3.6 Multiplica vectores de dimensión 1x2 y 2x1 ($\underline{\widehat{H}}_{ML}(k) = \underline{R}(k)\underline{T}(k)$)

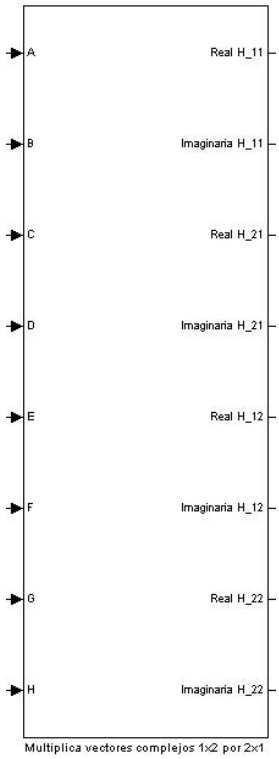


Figura 5.21: Multiplica Vectores complejos 1x2 por 2x1

En esta etapa del diseño (figura (5.21)), se desempeña el algoritmo de estimación de canal ML-TF. Este algoritmo consiste, en una multiplicación de dos vectores de elementos complejos de dimensiones 1x2 y 2x1, tal y como se define en la expresión (5.12). Esta multiplicación se va realizando para cada subportadora, dando como resultado una matriz 2x2 donde se tienen los coeficientes de canal en la subportadora que se está tratando en cada instante. Concretamente, $\underline{\underline{R}}(k)$ es un vector de dimensión 1x2 donde se tiene el valor del preámbulo de cada antena receptora para la subportadora "k" y $\underline{\underline{T}}(k)$ es un vector de dimensión 2x1 que contiene la fila de la matriz $\underline{\underline{T}}$ anteriormente comentada.

Esta etapa recibe la información del preámbulo en serie, de manera se recibe la información de cada subportadora secuencialmente. Por ejemplo, en un primer lugar se recibe $\underline{\underline{R}}(0)$, en un segundo lugar se recibe $\underline{\underline{R}}(1)$, y así sucesivamente hasta llegar a la información del preámbulo en la última subportadora, es decir, $\underline{\underline{R}}(255)$.

La llegada de los datos del preámbulo a esta etapa, está sincronizada con la lectura de los datos de la matriz $\underline{\underline{T}}$ (ver etapa anterior). Así, por ejemplo, cuando se lee la fila cero de la matriz $\underline{\underline{T}}$ (o lo que es lo mismo, leer la primera dirección de memoria de las memorias ROM de a etapa 5), llega el preámbulo en la portadora cero.

Como observamos en la figura (5.21), como entradas a esta etapa, tenemos 8 líneas. Las 4 últimas líneas, están destinadas a leer de las memorias de la etapa anteriormente descrita ($\underline{\underline{T}}(k)$). Por otro lado, las 4 primeras líneas hacen referencia a las partes reales e imaginarias del preámbulo recibido en cada antena ($\underline{\underline{R}}(k)$). Según la figura (5.21), la expresión de $\underline{\underline{R}}(k)$ quedaría:

$$\underline{\underline{R}}(k) = \begin{bmatrix} A + Bj \\ C + Dj \end{bmatrix} \qquad (5.13)$$

y la expresión de $\underline{\underline{T}}(k)$ quedaría:

$$\underline{\underline{T}}(k) = [E + Fj \quad G + Hj] \qquad (5.14)$$

Aplicando las expresiones (5.13) y (5.14) a (5.12), la multiplicación de vectores se realiza de la siguiente manera:

$$\underline{\underline{\hat{H}}}_{ML}(k) = \underline{\underline{R}}(k)\underline{\underline{T}}(k) = \begin{bmatrix} A + Bj \\ C + Dj \end{bmatrix} \cdot [E + Fj \quad G + Hj] \rightarrow$$

$$\rightarrow \underline{\underline{\hat{H}}}_{ML}(k) =$$

$$= \begin{bmatrix} (AE - BF) + (AF + EB)j & (AG - BH) + (AH + BG)j \\ (CE - DF) + (CF + ED)j & (CG - DH) + (CH + DG)j \end{bmatrix} \qquad (5.15)$$

La expresión (5.15), muestra las operaciones que hay que hacer para cada subportadora, es decir, 4 productos complejos. Cada producto complejo supone 4 multiplicaciones, una resta y una suma. Por tanto, la expresión (5.15) implica hacer 16 multiplicaciones, 4 sumas y 4 restas.

Esta etapa, realiza las operaciones de la expresión (5.15). Para ello, conecta adecuadamente diversos bloques que describen multiplicadores, sumadores y restadores. Todos estos bloques son lógica combinacional, lo que permite que la estimación de canal en una subportadora “k”, se realice en un solo ciclo de reloj. El conexionado de los distintos bloques que forman esta etapa, se ilustra en la figura (5.22).

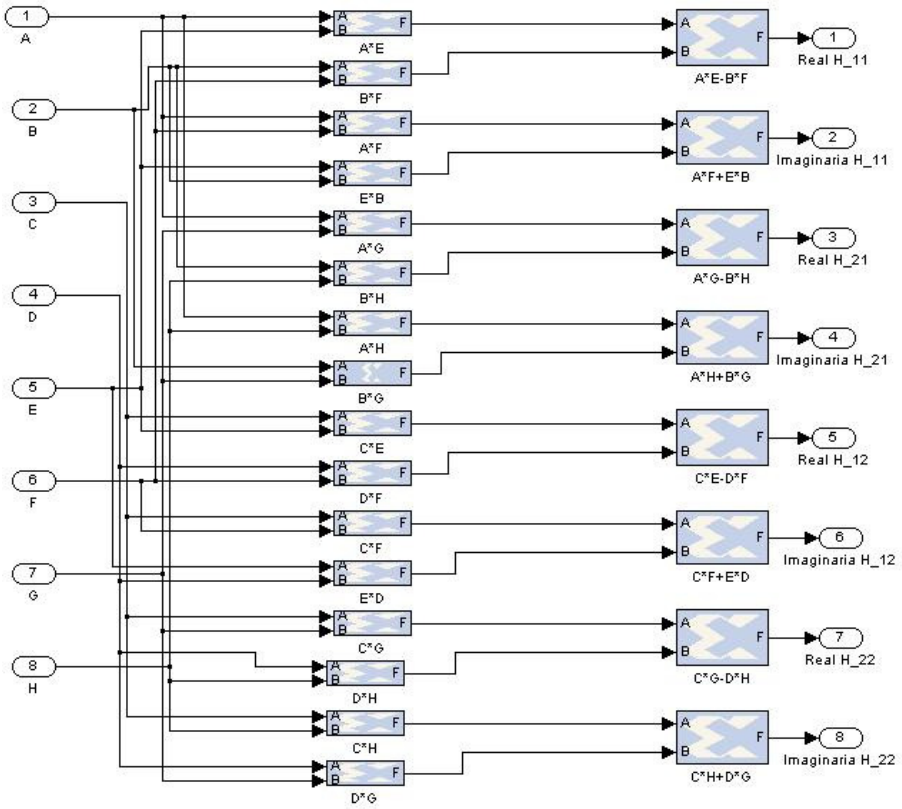


Figura 5.22: Interior del subsistema "Multiplica Vectores complejos 1x2 por 2x1"

Los bloques multiplicadores, sumadores y restadores de esta etapa, están modelados mediante bloques "***Black Box***", a los cuales se les asociará distintos códigos VHDL. Los códigos VHDL que modelan los multiplicadores, sumadores y restadores, los podemos encontrar en el anexo B.

En las figuras (5.23), (5.24) y (5.25), se ilustra el cuadro de configuración de los distintos bloques "***Black Box***" de esta etapa (multiplicadores, sumadores y restadores).

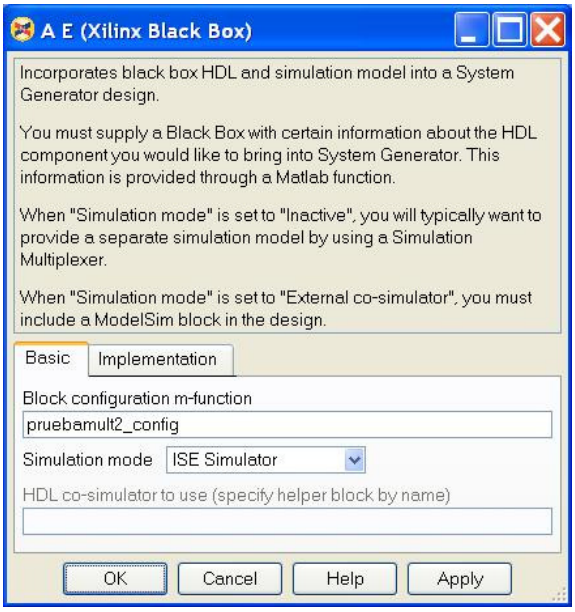


*Figura 5.23: Cuadro de configuración del bloque Black Box (bloque multiplicador "A*E")*

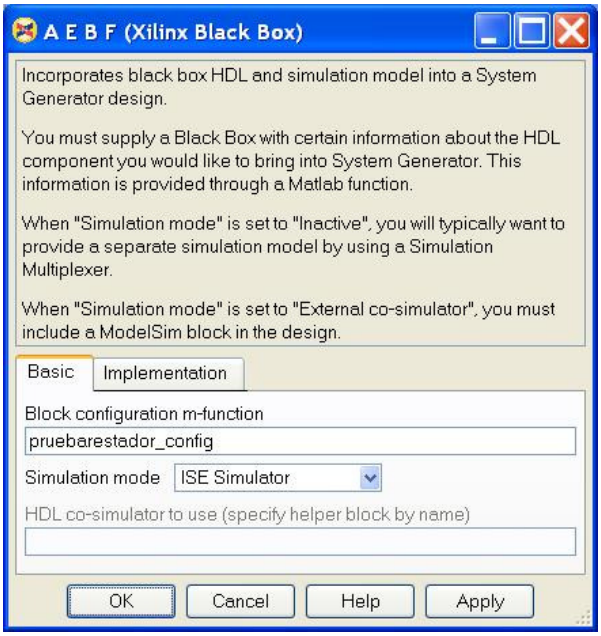


*Figura 5.24: Cuadro de configuración del bloque Black Box (bloque restador "A*E-B*F")*

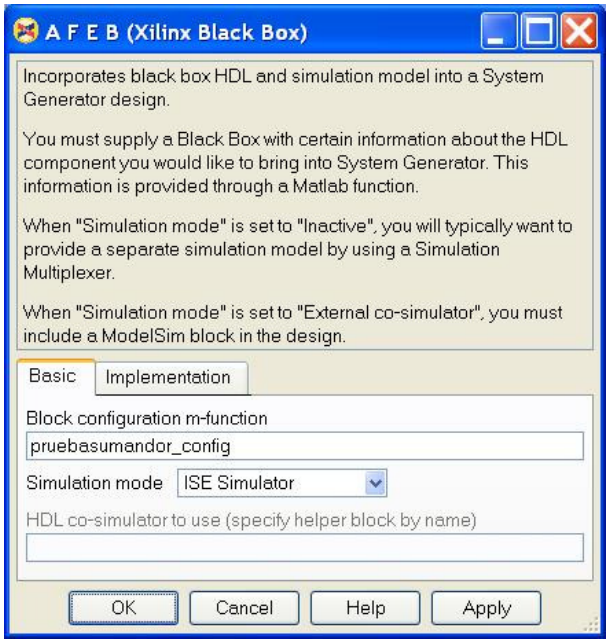


*Figura 5.25: Cuadro de configuración del bloque Black Box (bloque sumador "A*F+E*B")*

Las salidas de esta etapa, son las partes reales e imaginarias de los coeficientes de canal estimados en el dominio de la frecuencia, tal y como podemos observar en la figura (5.22).

3.7 Interpolación Lineal

El algoritmo de estimación de canal ML-TF, no estima el canal en todas las subportadoras ya que los preámbulos transmitidos no contienen información en todas las frecuencias. Habrá canales que estarán estimados en las subportadoras pares y otros que lo estarán en las subportadoras impares. Por ello, es necesario realizar una operación de interpolación para estimar el valor del canal en las subportadoras donde no se ha estimado.

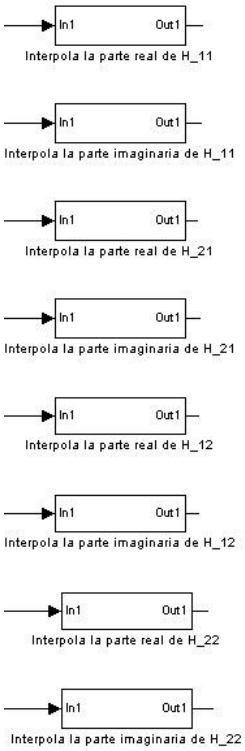


Figura 5.26: Etapa de interpolación lineal

Esta etapa (figura (5.26)) realizará una interpolación lineal en el dominio de la frecuencia de los canales estimados en la etapa anterior. La interpolación lineal se realizará de manera independiente sobre las partes reales e imaginarias de los canales estimados en la etapa anterior, tal y como se muestra en la figura (5.26). Cada línea de salida de la etapa anterior, se conecta con un subsistema que realiza la interpolación lineal. Por ejemplo, la salida "*Real H_11*" se conecta con el subsistema identificado como "*Interpola parte real de H_11*".

La interpolación lineal de una secuencia de dos muestras entre las que se intercala una muestra nula, se calcula sumando el valor de esas dos muestras y dividiéndolo entre 2. Esto lo podemos ilustrar en el ejemplo de la figura (5.27). En ella, observamos que tras la interpolación lineal, la muestra nula ha sido sustituida por una muestra que tiene un valor medio entre las dos muestras iniciales. En este caso, tenemos una muestra con valor 3 y otra con valor 1, por lo que tras la interpolación se genera una muestra de valor $(3+1)/2 = 2$ en el instante donde estaba la muestra nula.

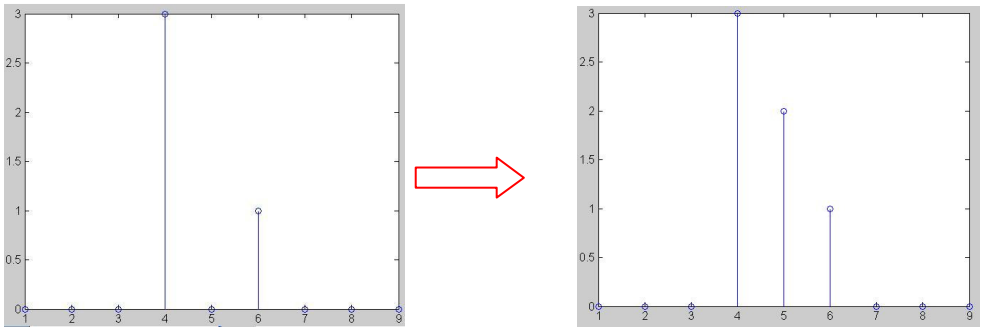


Figura 5.27: Ejemplo de interpolación lineal

El caso que nos ocupa, lo podemos aplicar al ejemplo sencillo anteriormente descrito. Al igual que sucede en el ejemplo, los canales estimados siguen una secuencia en la que entre cada par de muestras (o subportadoras), se intercala una muestra (o subportadora) nula.

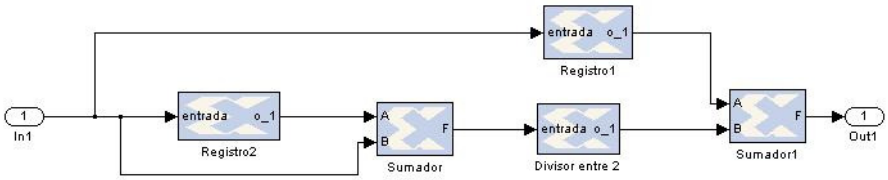


Figura 5.28: Interior del subsistema "Interpola parte real H_xx

Esta etapa (figura (5.26)), está formada por una serie de subsistemas que realizan una interpolación lineal como la mostrada en el ejemplo. Cada subsistema desdobla la señal de entrada en tres líneas, tal y como se muestra en la figura (5.28). Una de las líneas está retardada dos ciclos de reloj con respecto a la señal de entrada. Esta señal retardada dos ciclos de reloj (*bloque registro2*), se suma con la señal de entrada (*bloque sumador*) y se divide entre dos para obtener una señal que nos proporcionará los valores de la interpolación. Por otra parte, la señal de entrada esta retardada un ciclo de reloj (*bloque registro1*) con respecto a la señal que proporciona los valores de la interpolación para que al sumar estas dos señales (*bloque sumador2*) se rellenen las portadoras nulas con los valores de la interpolación, dando como resultado el la señal que representa el canal interpolado. Este proceso que acabamos de describir, se puede ilustrar en la figura (5.29), donde podemos ver que la señal a la salida del bloque es la señal de entrada interpolada.

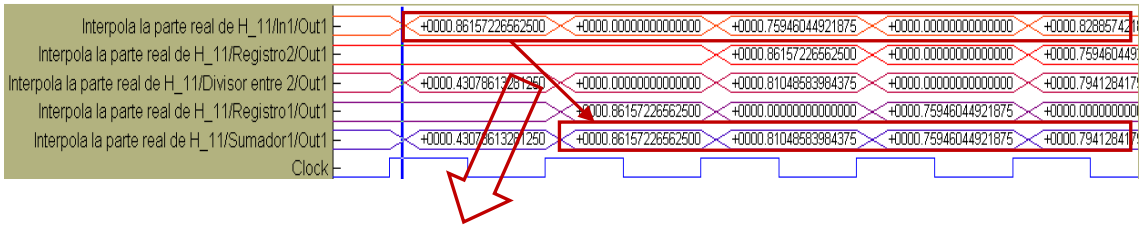


Figura 5.29: Señales del Interior del subsistema "Interpola parte real H_11"

El subsistema de interpolación mostrado en la figura (5.28), está compuesto por bloques "***Black Box***" que modelan registros de desplazamiento y sumadores. Los códigos VHDL asociados a estos bloques los podemos encontrar en el anexo B.

En las figuras (5.30), (5.31), (5.32) y (5.33), se ilustran los cuadros de configuración de los bloques "***Black Box***" empleados en la interpolación.

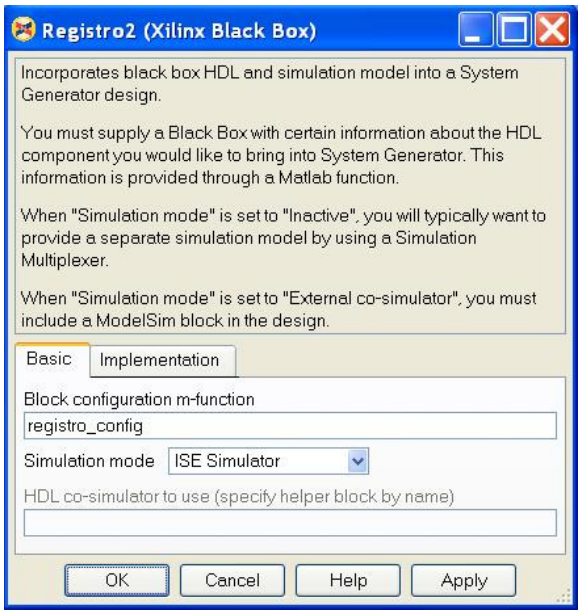


Figura 5.30: Cuadro de configuración del bloque Black Box (bloque registro de desplazamiento dos ciclos de reloj)

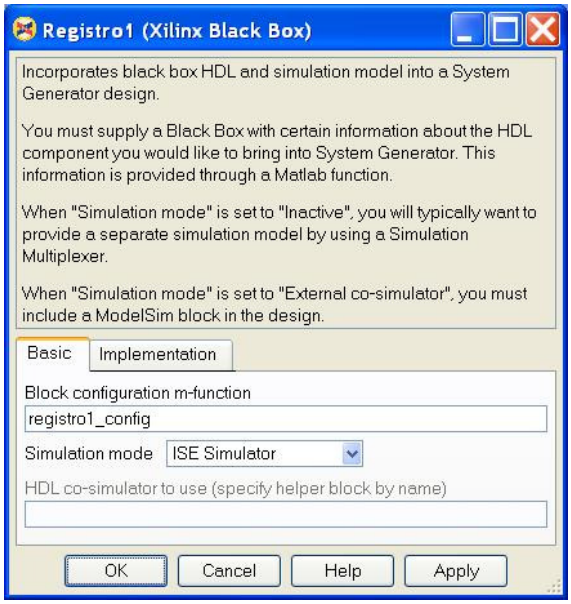


Figura 5.31: Cuadro de configuración del bloque Black Box (bloque registro de desplazamiento ciclo de reloj)

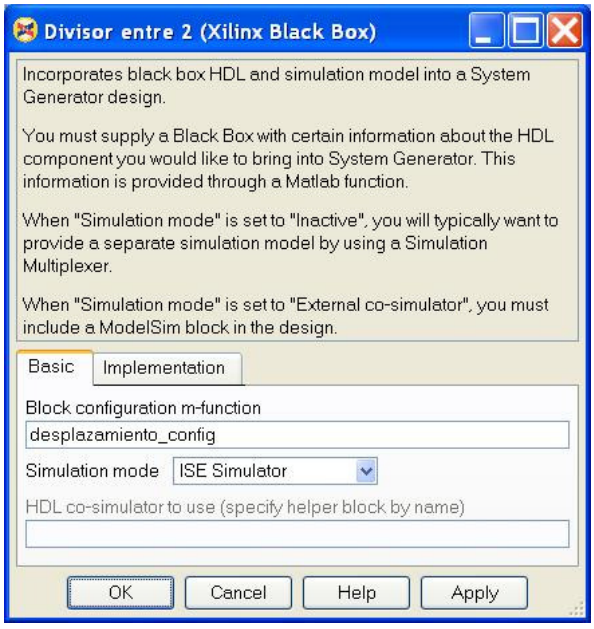


Figura 5.32: Cuadro de configuración del bloque Black Box (bloque divisor entre 2)

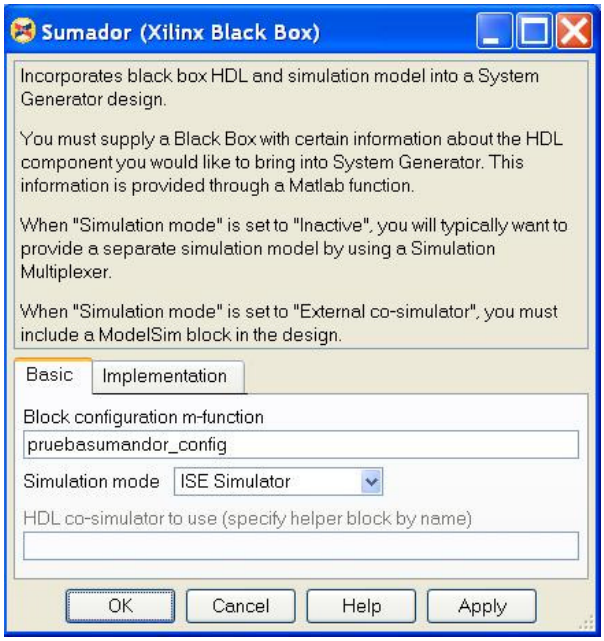


Figura 5.33: Cuadro de configuración del bloque Black Box (bloque sumador)

4. Validación del Hardware diseñado

Para comprobar que el hardware diseñado para el algoritmo de estimación de canal ML-TF funciona correctamente, validaremos tal hardware en los puntos mostrados en la figura (5.34). Los resultados obtenidos en esos puntos serán mostrados mediante el empleo del bloque "***Wavescope***". Los resultados obtenidos de la simulación del modelo Simulink®, los cuales se mostrarán en la ventana de visualización de "***Wavescope***", serán comparados con los resultados que ofrece una función de Matlab® que simula el algoritmo ML-TF. La función de Matlab® que simula el algoritmo de estimación de canal ML-TF es "*MLTF_offset.m*", la cual podemos encontrar en el anexo A.

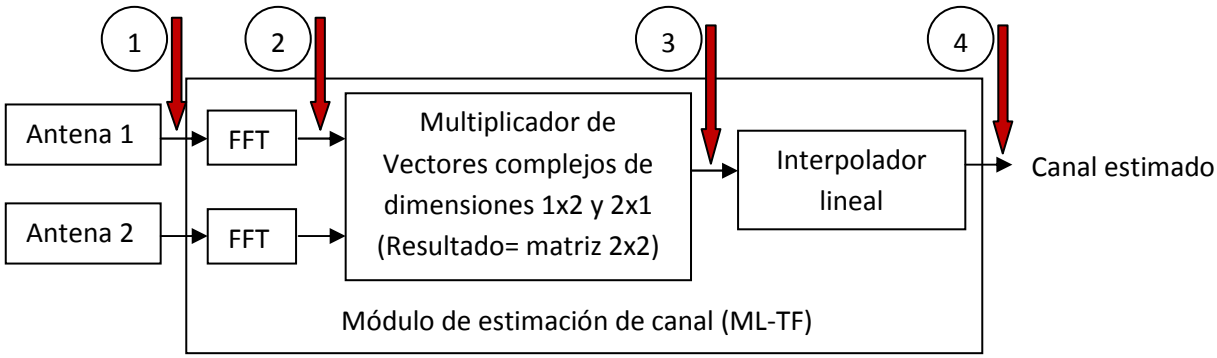


Figura 5.34: Puntos de validación del hardware diseñado

Los puntos a evaluar según la figura (5.34) son:

1) Entrada de datos
2) Salida del subsistema "*Hace la FFT de la señal recibida en el tiempo*"
3) Salida del subsistema "*Multiplica vectores de dimensión 1x2 y 2x1*"
4) Salida de los subsistemas "*Interpola parte real/imaginaria de H_xx*"

4.1 Entrada de datos

En este apartado compararemos el preámbulo recibido simulado en Matlab® y el preámbulo recibido que se obtiene de la simulación en Simulink®. En la figura (5.35), se ilustra el comienzo del preámbulo en ambas simulaciones para la antena 1 y, en la figura (5.36), se ilustra lo mismo pero referido a la antena 2.

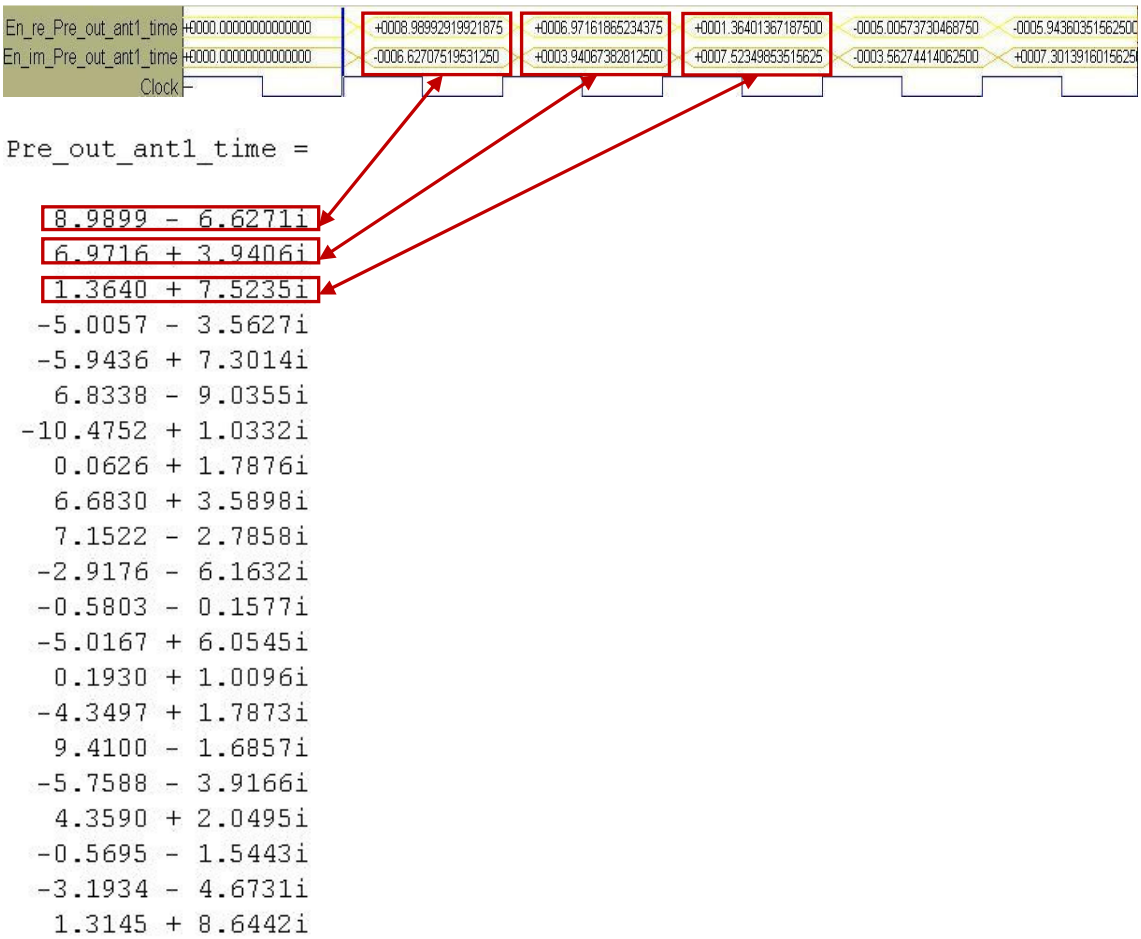
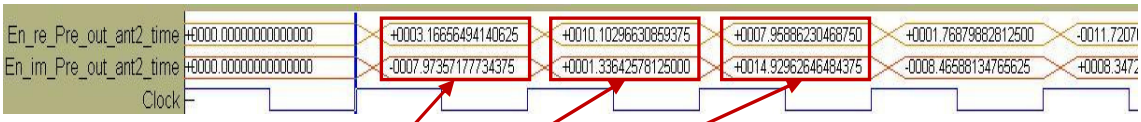


Figura 5.35: Preámbulo recibido en el dominio del tiempo para la antena receptora 1(comparativa entre Matlab® y Simulink®)



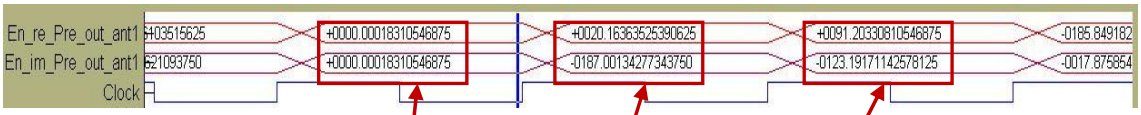

```
Pre_out_ant2_time =

  3.1666 - 7.9736i
 10.1030 + 1.3364i
  7.9589 +14.9297i
  1.7688 - 8.4659i
-11.7208 + 8.3473i
  8.4775 - 7.9987i
 -8.1973 - 2.2685i
 -7.2933 + 3.6353i
  4.8771 + 5.0343i
 10.2493 - 1.7424i
 -7.4863 - 6.4995i
  1.5559 - 6.6451i
  5.5488 + 7.8087i
  1.3737 + 6.1486i
 -9.7993 + 4.5439i
  5.6597 -10.6665i
-13.6202 + 1.8660i
 15.1356 - 0.1083i
 -7.9487 - 5.0531i
 -3.9604 + 2.6629i
 17.2408 + 9.8890i
```

Figura 5.36: Preámbulo recibido en el dominio del tiempo para la antena receptora 2(comparativa entre Matlab® y Simulink®)

4.2 Salida del subsistema "Hace la FFT de la señal recibida en el tiempo"

En esta etapa del diseño, vamos a comprobar que los bloques "***FFT v3_1***" hacen la operación deseada. En la figura (5.37), se ilustra la parte real e imaginaria de la FFT realizada por los bloques "***FFT v3_1***" en el modelo Simulink® para la antena 1 y lo que se obtiene de la simulación de Matlab® para la antena 1. Se debe obtener el preámbulo recibido en la antena 1 en el dominio de la frecuencia. En la figura (5.38), se ilustra lo mismo que en la figura (5.37) pero referido a la antena 2.

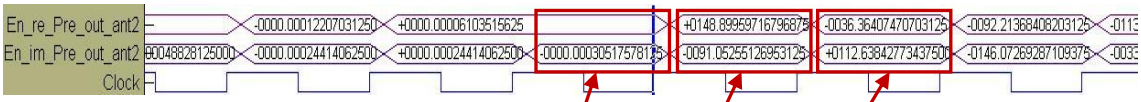



```
Pre_out_ant1 =

  1.0e+002 *

   0.0000 - 0.0000i
  -0.0000 - 0.0000i
         0
  -0.0000 - 0.0000i
  -0.0000 - 0.0000i
   0.0000 + 0.0000i
  -0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 - 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
  -0.0000 + 0.0000i
  -0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 - 0.0000i
  -0.0000 - 0.0000i
   0.0000 - 0.0000i
  -0.0000 + 0.0000i
  -0.0000 - 0.0000i
  -0.0000 + 0.0000i
  -0.0000 - 0.0000i
   0.0000 - 0.0000i
   0.0000 + 0.0000i
   0.0000 - 0.0000i
  -0.0000 + 0.0000i
  -0.0000 - 0.0000i
   0.2016 - 1.8700i
   0.9120 - 1.2319i
  -1.8585 - 0.1788i
   1.2138 + 0.9241i
   0.1567 - 1.8450i
   0.9367 - 1.1960i
  -0.1357 + 1.8297i
   0.9499 - 1.1785i
   0.1158 - 1.8125i
   0.9636 - 1.1613i
   1.7937 + 0.0973i
  -1.1445 - 0.9779i
   1.7733 + 0.0803i
```

Figura 5.37: Preámbulo recibido en el dominio de la frecuencia para la antena receptora 1(comparativa entre Matlab® y Simulink®)




```
Pre_out_ant2 =

  1.0e+002 *

   0.0000 + 0.0000i
   0.0000 + 0.0000i
        0
   0.0000 + 0.0000i
   0.0000 - 0.0000i
  -0.0000 - 0.0000i
  -0.0000 + 0.0000i
  -0.0000 - 0.0000i
  -0.0000 + 0.0000i
  -0.0000 + 0.0000i
   0.0000 + 0.0000i
  -0.0000 - 0.0000i
   0.0000 + 0.0000i
   0.0000 - 0.0000i
  -0.0000 - 0.0000i
  -0.0000 - 0.0000i
   0.0000 - 0.0000i
   0.0000 + 0.0000i
   0.0000 - 0.0000i
   0.0000 - 0.0000i
   0.0000 - 0.0000i
   0.0000 - 0.0000i
   0.0000 - 0.0000i
  -0.0000 + 0.0000i
  -0.0000 + 0.0000i
  -0.0000 + 0.0000i
  -0.0000 - 0.0000i
   0.0000 + 0.0000i
   1.4890 - 0.9105i
  -0.3636 + 1.1264i
  -0.9221 - 1.4607i
  -1.1301 - 0.3359i
   1.4320 - 0.9312i
  -0.3078 + 1.1300i
  -1.4029 + 0.9376i
  -0.2798 + 1.1260i
   1.3739 - 0.9415i
  -0.2522 + 1.1180i
   0.9429 + 1.3450i
   1.1063 + 0.2256i
   0.9418 + 1.3166i
  -1.0908 - 0.2003i
  -0.9383 - 1.2888i
  -1.0718 - 0.1768i
  -0.9325 - 1.2620i
   1.0494 + 0.1555i
```

Figura 5.38: Preámbulo recibido en el dominio de la frecuencia para la antena receptora 2(comparativa entre Matlab® y Simulink®)

En las capturas de las figuras (5.37) y (5.38), no aparece toda la banda de guarda que hay al comienzo de los preámbulos en el dominio de la frecuencia, por lo que se han capturado las imágenes a partir de las portadoras con información.

4.3 Salida del subsistema "Multiplica vectores de dimensión 1x2 y 2x1"

En esta etapa del diseño, vamos a comprobar que el subsistema encargado de realizar la estimación de canal en cada subportadora, realiza las operaciones correctamente. Para ello, hemos comparado los resultados obtenidos de la simulación de canal en Matlab®, con los obtenidos de la simulación del modelo Simulink®. En las figuras (5.39), (5.40), (5.41) y (5.42), se ilustra esta comparación, para los cuatro canales del canal 2x2 MIMO, mostrando solo las primeras subportadoras con información (excluimos las subportadoras de las bandas de guarda).

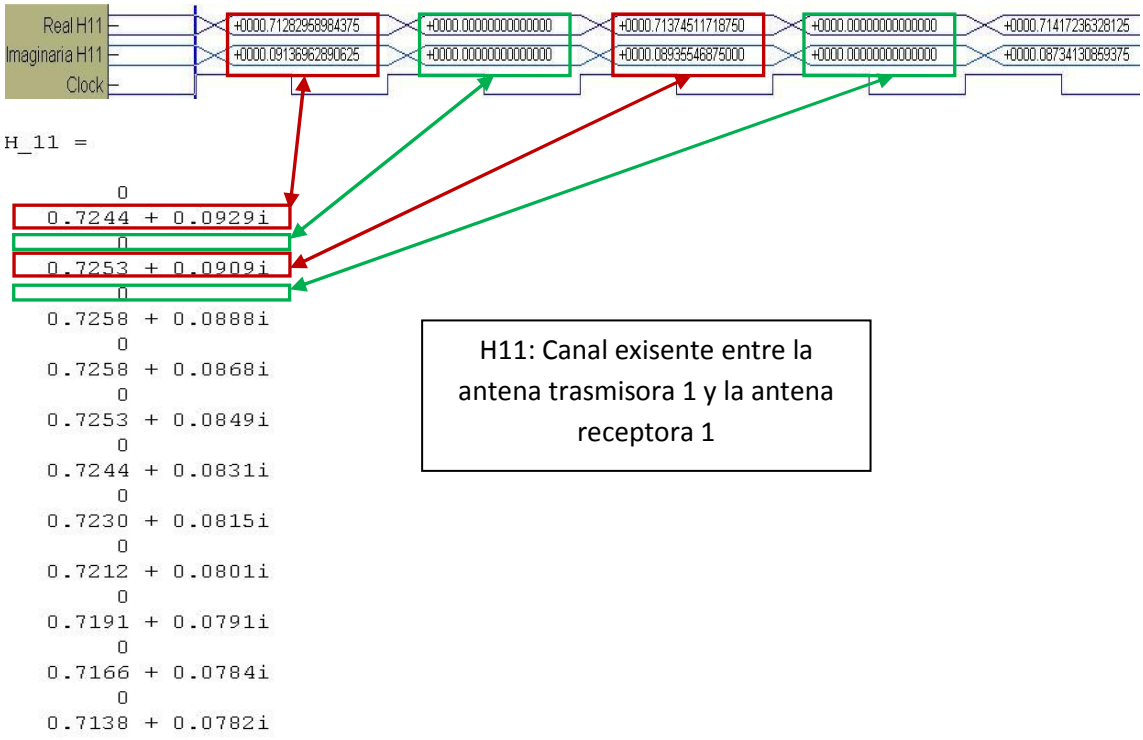


Figura 5.39: Canal estimado H11 (comparativa entre Matlab® y Simulink®)

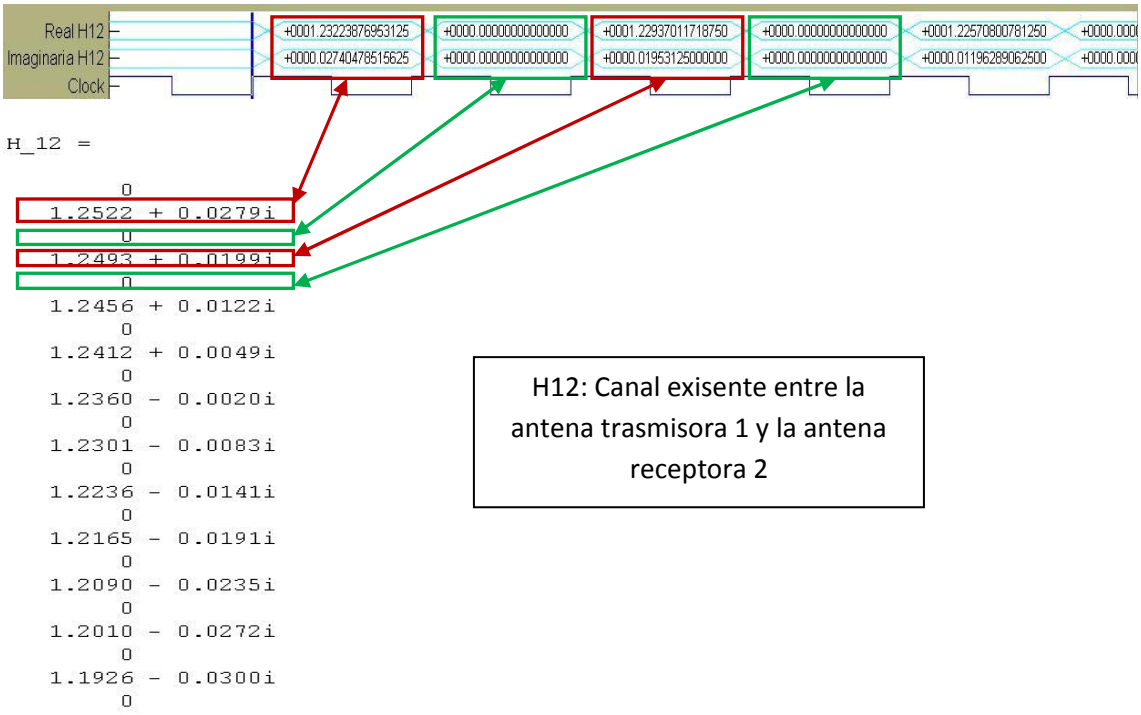


Figura 5.40: Canal estimado H12 (comparativa entre Matlab® y Simulink®)

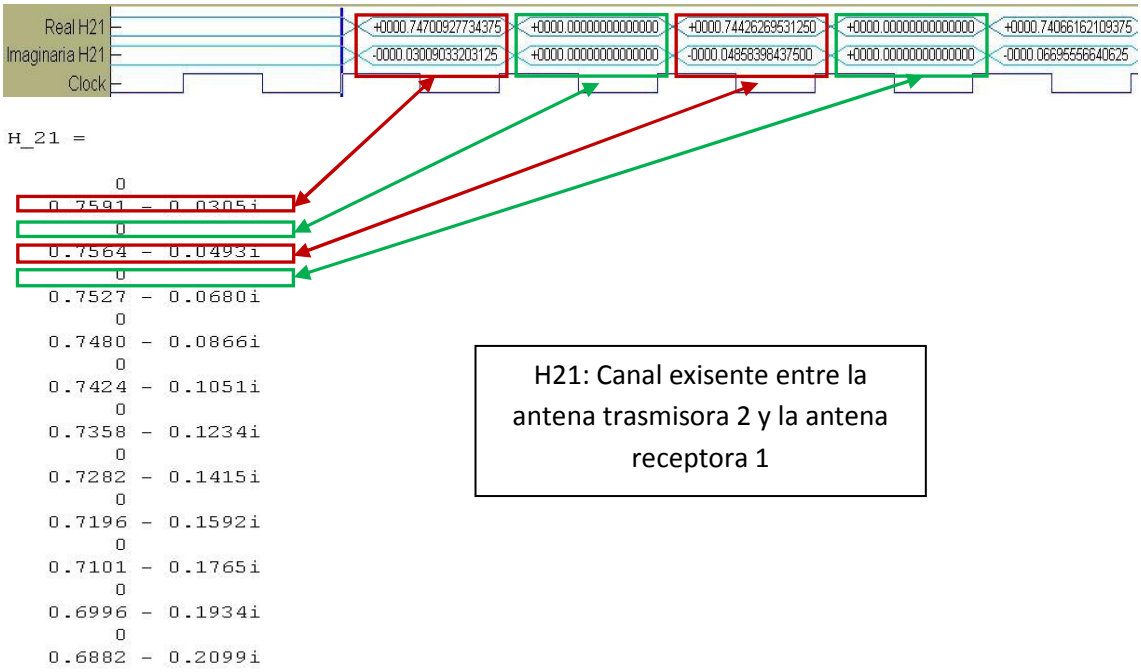


Figura 5.41: Canal estimado H21 (comparativa entre Matlab® y Simulink®)

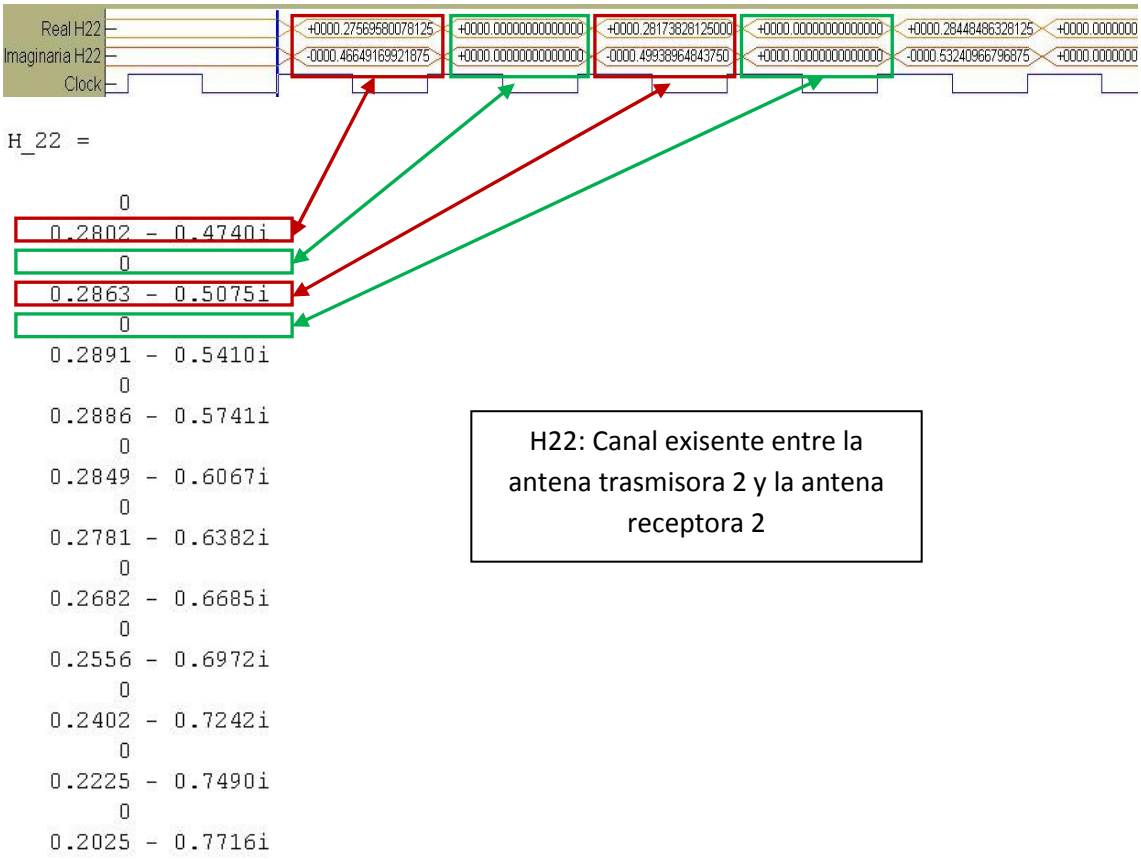


Figura 5.42: Canal estimado H22 (comparativa entre Matlab® y Simulink®)

➔ Observaciones.

1) Notar como los datos proporcionados por la simulación del modelo Simulink®, no son exactamente los mismos que se obtienen en la simulación de Matlab®. Esto se debe a que Matlab® realiza cálculos con datos tipo "*Double*" y el modelo Simulink® está empleando datos tipo "*signed*" de 24 bits con punto binario en el bit 14. La precisión de Matlab® es mayor, y por tanto los resultados son más exactos que los proporcionados por Simulink®.
2) Se puede apreciar en las capturas como no se estima el canal en todas las subportadoras, por lo que será necesario la interpolación lineal para estimar el resto de frecuencias.

4.4 Salida de los subsistemas "Interpola parte real/imaginaria de H_xx"

En este apartado, trataremos de ver si la etapa de interpolación lineal realiza tal operación correctamente. Para comprobar que esta etapa realiza bien su función, no necesitaremos comparar su salida con los resultados de la función de Matlab® que simula el algoritmo ML-TF, si no ver si la señal de entrada a esta etapa está interpolada a la salida. Así, solo bastará con comprobar que uno de los canales del canal 2x2 MIMO se ha interpolado correctamente, ya que el subsistema que realiza la interpolación es el mismo para todos los canales (ver figura (5.26)). Por tanto, en la figura (5.43), se ilustra cómo se realiza tal interpolación lineal para el canal H_11.

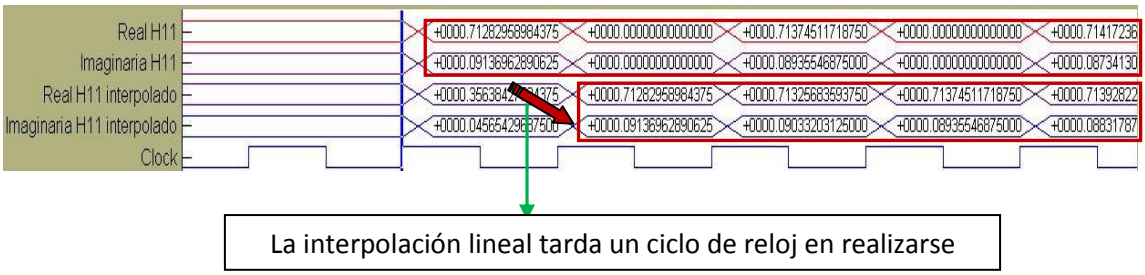


Figura 5.43: Interpolación lineal para el canal H11

5. Resultados

En este apartado mostraremos los resultados de la estimación de canal ML-TF. Para ello, importaremos a Matlab® los resultados de la simulación del modelo Simulink®, mediante bloques "***To Workspace***", los cuales generarán vectores en el "*workspace*" de Matlab® que contendrán los resultados de la simulación. Estos vectores serán pasados como entradas a la función de Matlab® "*representa.m*", la cual será la encargada de representar los resultados de la estimación de canal. Esta función de Matlab® la podemos encontrar en el Anexo A.

Para comprobar que el resultado final de la estimación es correcto, también, se mostrará el resultado de la simulación de Matlab® (función de Matlab® "*MLTF_offset.m*", ver anexo A) junto con el resultado de la simulación del modelo Simulink®. Así, en la figura (5.44), se muestra el resultado de la simulación de Matlab® del algoritmo ML-TF y, en la figura (5.45), se muestra el

resultado de la simulación del modelo Simulink® que modela el algoritmo ML-TF.

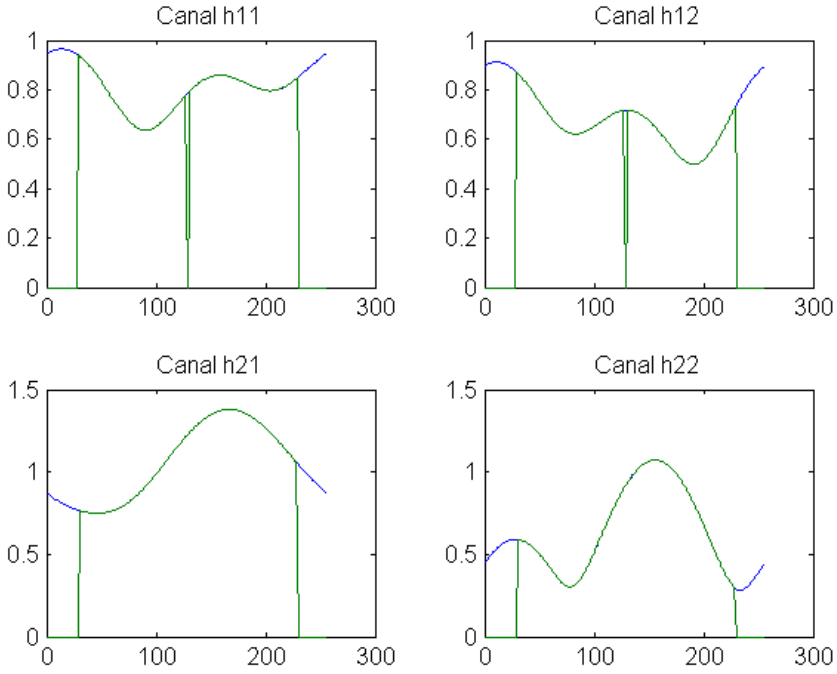


Figura 5.44: Resultado de la estimación de canal de la simulación de Matlab®

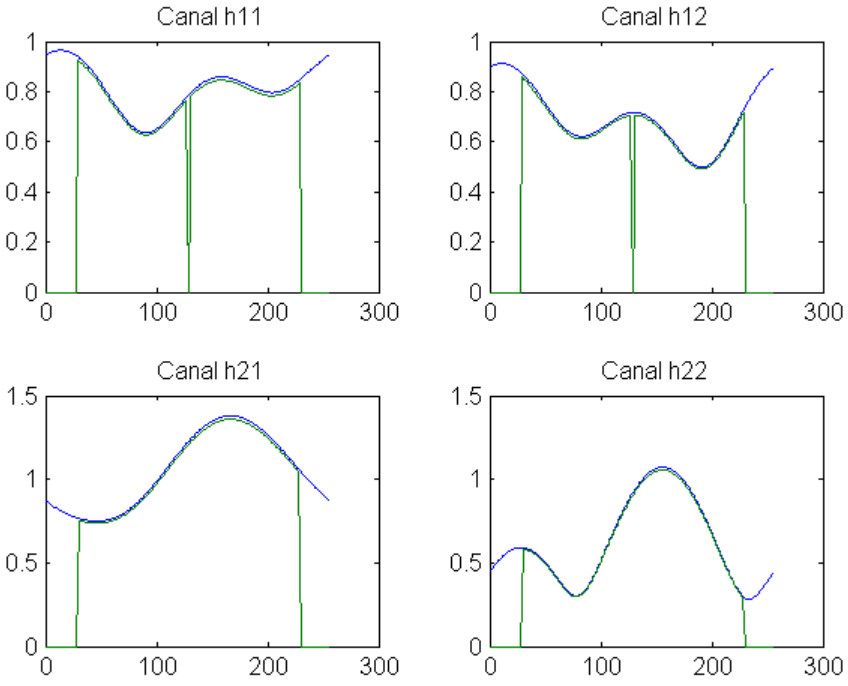


Figura 5.45: Resultado de la estimación de canal realizada por el hardware diseñado

Podemos ver en las figuras anteriores, que los resultados son prácticamente el mismo. La pequeña diferencia que se puede apreciar, es debida a que el hardware diseñado tiene un número limitado de bits y la simulación de Matlab® emplea mayor número de bits para tratar los datos.

Capítulo 6

Algoritmos de Sincronización Tiempo-Frecuencia

En el presente capítulo, trataremos de describir los algoritmos de sincronización en frecuencia y sincronización temporal que describiremos en hardware, a través de la utilización de las herramientas anteriormente citadas.

Para situar al lector en el ámbito que trataremos, en un primer lugar, estudiaremos, de una manera muy breve, en qué se basa la sincronización frecuencial y la sincronización temporal, para posteriormente, hacer una recopilación de distintos algoritmos de sincronización temporal y sincronización en frecuencia que podemos encontrar en la literatura. Así, se pretende dar una visión del estado del arte, en lo que a estos tipos de algoritmos se refiere.

En un segundo lugar, nos centraremos en describir de manera, tanto teórica como práctica, los algoritmos concretos que utilizaremos en la sincronización tiempo-frecuencia. Se deberá comprobar el correcto funcionamiento de los pertinentes algoritmos descritos en hardware. Así, una vez desarrollado el hardware relativo a la sincronización tiempo-frecuencia, se validará tal hardware, en distintas zonas del diseño, para comprobar que funciona correctamente.

1. Estado del Arte

Los sistemas OFDM son muy sensibles a fallos en la sincronización de las señales recibidas. La no sincronización de los elementos del sistema puede suponer una considerable degradación de la calidad del sistema. Estos sistemas necesitan una buena sincronización tiempo-frecuencia, de lo contrario, se produce una perdida de ortoganalidad entre subportadoras que provoca la aparición de interferencia entre símbolos. Hay que diferenciar entre fallos de sincronización temporal y fallos de sincronización entre los osciladores de transmisores y receptores.

Los fallos en la sincronización temporal son propiciados por el desconocimiento del momento de llegada de la trama, mientras que los fallos de sincronización en frecuencia son debidos a que la señal recibida presenta una desvición en la portotadora de radio frecuencia.

Estos efectos se pueden modelar sigún la expresión [34]:

$$r(n) = \big(s(n-v) * h(n)\big) \cdot e^{\frac{j2\pi\varepsilon n}{N}} \quad (6.1)$$

donde $r(n)$ es la señal recibida, $s(n-v)$ es la señal transmitida tras un cierto instante de tiempo v que representa la desviación temporal sufrida y $e^{\frac{j2\pi\varepsilon n}{N}}$ representa la desviación en frecuencia sufrida por la señal. El parámetro ε de la exponencial compleja se conoce como desviación en frecuencia normalizada, el cual se define:

$$\varepsilon = \frac{f_\varepsilon}{\Delta f} \quad (6.2)$$

donde f_ε es la desviación en frecuencia y Δf es la separación entre subportadoras.

Los algortimos de sincronización tiempo-frecuencia tratarán de estimar los parámetros v y ε, para tratar de corregirlos y evitar que la calidad del sistema se degrade. Existen infinidad de métodos en la literatura que estiman el desplazamiento temporal y frecuencial, por lo que para contextualizar al lector en el ámbito que trataremos, citaremos algunos algoritmos presentes en la literatura.

En [26] se realiza la estimación del desplazamiento frecuencial mediante el empleo de un estimador de máxima verosimilitud en el dominio del tiempo. Trata de aumentar el rango de estimación del desplazamiento en frecuencia a

través de el empleo de preámbulos mas cortos, sin embargo, esto tiene un límite por que hay menos muestras para estimar.

En [27] se propone un esquema eficiente de autocorrelación eficiente para la estimación simultánea del desplazamiento frecuencial y temporal. Reducen la complejidad del Hardware mediante la reducción del número de bits empleados en la estimación y un esquema de correlación basada en una técnica de multiplexacón en el tiempo.

En [28] se propone un esquema eficiente para la estimación del desplazamiento temporal y frecuencial mediante la separación de las operaciones de correlación que llevan asociada estas estimaciones. Para determoninar el desplazamiento en frecuencia propone un método basado en la correlacion cruzada.

En [29], se trata de estimar el el desplazamiento temporal a través de un algortimo que se basa en la correlación del prefijo cíclico.

En [30], se estima conjuntamente el desplazamiento temporal y el frecuencial conjuntamente a través de un método de prueba y error. Sin embargo, este método tiene una alta carga computacional.

En [31], se propone un método para la determinación del desplazamiento temporal basado en el empleo de una métrica que posee un intervalo de incertidumbre. Sin embargo, en [32] se propone un método que reduce esta zona de incertidumbre, la cual determina cuando se recibe la trama.

2. Explicación Teórica del algoritmo de Sincronización en frecuencia

En el presente apartado, se explicará teóricamente el algoritmo empleado en la estimación del desplazamiento en frecuencia, el cual será descrito en hardware posteriormente. Concretamente, se ha elegido el algoritmo presentado en la referencia bibliográfica [1]. El algoritmo de estimación del desplazamiento frecuencial presentado en [1] está ideado para un sistema de comunicaciones distinto al que estamos tratando en el presente proyecto, por lo que adecuaremos al sistema de comunicación que nos ocupa.

El sistema de comunicaciones ideado utiliza el preámbulo descrito en el apartado (2.1.3.1), por lo que este algoritmo se ha adaptado a tal preámbulo. Concretamente, una de las antenas emite un símbolo OFDM con información

en las subportadoras pares (deducido de la secuencia P_{EVEN}) y la otra un símbolo OFDM en las portadoras impares (deducido de la secuencia P_{ODD})

Este apartado tiene como objetivo, adaptar el algoritmo presentado en [2] para obtener un método de estimación y corrección del "*offset*" en frecuencia para sistemas MIMO-OFDM como el que estamos tratando.

Podemos desglosar este algoritmo adaptado a nuestro sistema en los siguientes pasos:

- El primer paso, a llevar a cabo, será la estimación del canal MIMO mediante el algoritmo ML-TF descrito en el capítulo anterior. Evidentemente, en esta primera estimación de canal no tendremos una buena estimación debido a la existencia de un desplazamiento en frecuencia (canal distorsionado). Sin embargo, supondremos que para este primer paso el desplazamiento frecuencial ha sido compensado.

- Una vez se ha hecho la primera estimación del canal en el dominio de la frecuencia, definimos las señales que se reciben en el dominio del tiempo. Distinguiremos entre:

  1) La señal que realmente llegan a la q-ésima antena receptora, $\underline{y_t}(q)$. La notación que emplearemos es la siguiente: $\underline{y_t}(q,1)$ es la primera mitad del preámbulo empleado en la sincronización en frecuencia en el dominio del tiempo, es decir, las 128 primeras muestras del símbolo OFDM que se envía como preámbulo. $\underline{y_t}(q,2)$ es exactamente lo mismo que $\underline{y_t}(q,1)$ pero referida a la segunda mitad del símbolo OFDM.

  2) Suponiendo que la primera estimación de canal es correcta y sabiendo como es el preámbulo que debería transmitirse, se puede obtener el preámbulo que debería recibirse si la primera estimación de canal fuese correcta.
     El preámbulo que debería llegar al receptor, suponiendo que la primera estimación de canal es correcta es:

     a) Primera mitad del preámbulo en el dominio del tiempo (128 muestras):

$$\underline{y_t}(q,1)' = \sum_{p=1}^{P}\sum_{l=0}^{L-1} \underline{\underline{H_l}}\,(q,p)\,\underline{x}_{t-k}(p) \qquad (6.3)$$

b) Segunda mitad del preámbulo en el dominio del tiempo (128 muestras):

$$\underline{y_t}(q,2)' = \sum_{p=1}^{P}\sum_{l=0}^{L-1} \underline{\underline{H_l}}\,(q,p)\,\underline{x}_{t+128-k}(p) \qquad (6.4)$$

donde $\underline{\underline{H_l}}(q,p)$ es la respuesta en el dominio del tiempo del canal estimado en el primer paso entre la p-ésima antena transmisora y la q-ésima antena receptora (l indica el numero de taps o caminos del canal ($l = 0, \dots, L-1$), que en nuestro caso será tres $L = 3$), $\underline{x}_{t-k}(p)$ hace referencia a la primera mitad del preámbulo, $\underline{x}_{t+128-k}(p)$ hace referencia a la segunda mitad del preámbulo, $\underline{y_t}(q,1)'$ hace referencia a la primera mitad del preámbulo que se debería recibir en la q-ésima antera receptora y $\underline{y_t}(q,2)'$ hace referencia a la segunda mitad del preámbulo que se debería recibir en la q-ésima antera receptora.

Las operaciones (6.3) y (6.4) están realizadas en el dominio del tiempo. Sin embargo, al realizar previamente la estimación de canal en el dominio de la frecuencia, haremos estas operaciones en el dominio de la frecuencia y al resultado obtenido, le aplicaremos una operación de IDFT para obtener su equivalente en el dominio del tiempo. El sistema de comunicación del presente proyecto, es un sistema MIMO 2x2, por lo que tenemos 4 canales ($H_{11}, H_{12}, H_{21}, H_{22}$). Así, para llegar al resultado marcado en las expresiones (6.3) y (6.4), procederemos de la siguiente manera:

$$\underline{Y}(q) = \underline{H_{1q}}\underline{x}_{P_{EVEN}} + \underline{H_{2q}}\underline{x}_{P_{ODD}} \qquad (6.5)$$

donde $\underline{H_{1q}}$ es un vector de 256 elementos (porque trabajamos con 256 subportadoras) donde se almacena la respuesta en frecuencia del canal (estimado en el primer paso) existente entre la antena transmisora 1 y la q-ésima antena receptora, $\underline{H_{2q}}$ es un vector de 256 elementos (porque trabajamos con 256 subportadoras) donde se almacena la respuesta en frecuencia del canal existente entre la

antena transmisora 3 y la q-ésima antena receptora, $\underline{x}_{P_{EVEN}}$ es un vector de 256 elementos donde se almacena el preámbulo emitido por la antena transmisora 1 en el dominio frecuencial, $\underline{x}_{P_{ODD}}$ es una vector de 256 elementos donde se almacena el preámbulo emitido por la antena transmisora 2 en el dominio frecuencial y $\underline{Y}(q)$ es el preámbulo recibido en la q-ésima antena receptora en el dominio de la frecuencia.

Particularizando la expresión (6.5) para la antena $q = 1$ y $q = 2$, se tiene:

$$\underline{Y}(1) = \underline{H}_{11}\underline{x}_{P_{EVEN}} + \underline{H}_{21}\underline{x}_{P_{ODD}} \qquad (6.6)$$

$$\underline{Y}(2) = \underline{H}_{12}\underline{x}_{P_{EVEN}} + \underline{H}_{22}\underline{x}_{P_{ODD}} \qquad (6.7)$$

Si realizamos una operación de IDFT de 256 puntos de la expresión (6.6) y dividimos la señal resultante en el dominio del tiempo en dos mitades de igual longitud, se tiene:

$$\underline{Y}(q) \rightarrow IDFT\ 256\ puntos \rightarrow \underline{y}_t(q)' \qquad (6.8)$$

Si dividimos en dividimos en dos mitades iguales el preámbulo recibido, se tiene el resultado de (6.3) y (6.4):

$$\underline{y}_t(q)' = \begin{cases} \underline{y}_t(q,1)', & t < 128 \\ \underline{y}_t(q,2)', & t \geq 128 \end{cases} \qquad (6.9)$$

- Ahora, una vez definidas las señales que van a intervenir en la estimación del "*offset*", procedemos a la estimación del mismo. Según, la referencia [2], se puede estimar el desplazamiento frecuencial por el cambio de fase entre $\underline{y}_t(q,1), \underline{y}_t(q,1)'^*$ y $\underline{y}_t(q,2), \underline{y}_t(q,2)'^*$. Así, se puede calcular un desplazamiento fraccional de frecuencia en la q-ésima antena receptora según la siguiente expresión:

$$\hat{\epsilon}_{frac_q} = angle\left(\sum_{t=0}^{127}\left(\underline{y}_t(q,1)\,\underline{y}_t(q,1)'^*\right)^*\left(\underline{y}_t(q,2)\,\underline{y}_t(q,2)'^*\right)\right) \qquad (6.10)$$

donde "*angle*" implixa calcular la fase del témino entre paréntesis.

Si se quiere reducir el efecto del ruido, se considerarán el resto de antenas receptoras. En nuestro caso, tenemos 2 antenas receptoras, por lo que la estimación del "*offset*" frecuencial queda:

$$\hat{\epsilon}_{frac} = angle\left(\sum_{t=0}^{127}\left(\underline{y}_t(1,1)\,\underline{y}_t(q,1)'^{*}\right)^{*}\left(\underline{y}_t(1,2)\,\underline{y}_t(q,2)'^{*}\right) + \sum_{t=0}^{127}\left(\underline{y}_t(2,1)\,\underline{y}_t(2,1)'^{*}\right)^{*}\left(\underline{y}_t(2,2)\,\underline{y}_t(2,2)'^{*}\right)\right) \qquad (6.11)$$

3. Explicación Teórica del algoritmo de Sincronización temporal

En el presente apartado, se explicará teóricamente el algoritmo empleado en la estimación del desplazamiento temporal, el cual será descrito en hardware posteriormente. Concretamente, se ha elegido el algoritmo presentado en la referencia bibliográfica [3], el cual está basado en la correlación de las dos mitades idénticas que posee el preámbulo enviado.

El sistema de comunicaciones ideado utiliza el preámbulo descrito en el apartado (2.1.3.1), el cual es el mismo que se propone en la referencia bibliográfica [3]. Como ya se vio en el aparatado (2.1.3.1) del capítulo 2, este preámbulo será transmitido por una sola antena de manera que para la sincronización temporal se tiene un sistema SISO.

El algoritmo de sincronización temporal busca detectar el comienzo de una trama mediante la búsqueda de un preámbulo con dos mitades idénticas en el dominio del tiempo, las cuales permanecerán idénticas después de pasar a través del canal. Sin embargo, el método descrito por Tae-Hwan Kim y In-Cheol Park en [3], trata de correlar dos fragmentos de 64 muestras idénticos, que se corresponden con la mitad el preámbulo siguiente (ver apartado (2.1.3.1)):

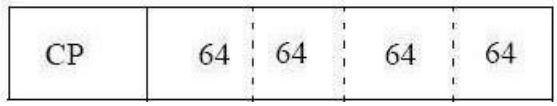


Figura 6.1: Preámbulo utilizado en la sincronización temporal [3]

Según [3], el instante de llegada de la trama se derivará de la siguiente métrica:

$$\mu_n = \frac{\left|\sum_{i=1}^{W} r_{n-i} \cdot r_{n-i-W}^{*}\right|^2}{\left|\sum_{i=1}^{W} |r_{n-i}|^2\right|^2} \qquad (6.12)$$

donde la parte del numerador $\sum_{i=1}^{W} r_{n-i} \cdot r_{n-i-W}^{*}$ realiza la correlación de los fragmentos idénticos del preámbulo de duración W (64 muestras en nuestro caso) y la parte del denominador $\sum_{i=1}^{W} |r_{n-i}|^2$ calcula la energía de un fragmento de preámbulo de duración W (64 muestras en nuestro caso). Para calcular la auto-correlación de los fragmentos de preámbulo, se define una ventana de duración 2W que se va desplazando a lo largo del tiempo. Así, r_{n-i} hace referencia a las muestras que hay en la primera mitad de la ventana y r_{n-i-W}^{*} hace referencia a las muestras que hay en la segunda mitad de la ventana. Cuando la ventana esté alineada temporalmente con el preámbulo se producirá un máximo en la auto-correlación y se detectará el instante de llegada de la trama.

Notar que se usa la métrica en lugar de la auto-correlación. Esto es debido a que el pico de auto-correlación puede ser muy alto, por lo que se utiliza la energía de un fragmento de preámbulo para normalizar el valor de la auto-correlación, tal y como podemos ver en la métrica (6.12).

La auto-correlación que se muestra en el numerador de la métrica (6.12), según [3], se puede escribir según la siguiente fórmula iterativa:

$$\Lambda_n = \sum_{i=1}^{W} r_{n-i} \cdot r_{n-i-W}^{*} = \Lambda_{n-1} + r_{n-1} \cdot r_{n-W}^{*} - r_{n-W} \cdot r_{n-2W+1}^{*} \qquad (6.13)$$

También, el término del denominador de la métrica (6.12) se puede expresar mediante la siguiente fórmula iterativa:

$$E_n = \sum_{i=1}^{W} |r_{n-i}|^2 = E_{n-1} + |r_{n-W}|^2 - |r_{n-1}|^2 \quad (6.14)$$

En el caso que nos ocupa, las fórmulas iterativas (6.13) y (6.14), requiere un "*buffer*" con una capacidad de 128 muestras (2W). Así, en la figura (6.1) se esquematiza cómo funciona el algoritmo. En ella, podemos observar la necesidad de un "*buffer*" circular de 128 muestras (hace de ventana de 2W muestras de duración) y como se definen los bloques para llevar a cabo las fórmulas iterativas anteriormente descritas. A medida que el preámbulo va llenando el "*buffer*" la auto-correlación definida en (6.13) va aumentando su valor.

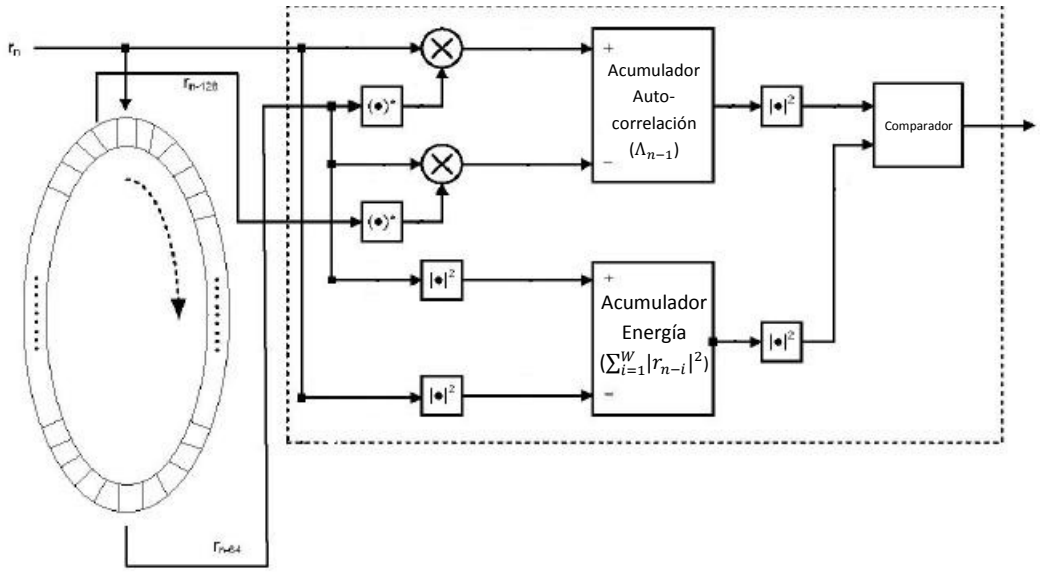


Figura 6.2: Algoritmo de sincronización temporal [3]

Para detectar el instante de llegada de la trama, la métrica (6.12) es comparada con un umbral, de manera que cuando la métrica (6.12) supera ese umbral se detecta el comienzo de la trama. Esto se muestra a continuación:

$$\frac{\left|\sum_{i=1}^{W} r_{n-i}\cdot r_{n-i-W}^{*}\right|^{2}}{\left|\sum_{i=1}^{W}|r_{n-i}|^{2}\right|^{2}} > th \rightarrow \left|\sum_{i=1}^{W} r_{n-i}\cdot r_{n-i-W}^{*}\right|^{2} - th\left|\sum_{i=1}^{W}|r_{n-i}|^{2}\right|^{2} > 0 \qquad (6.15)$$

Como podemos deducir de la expresión (6.15), determinar el instante de llegada de la trama se realiza con una simple comparación, tal y como se ilustra en la figura (6.1) donde aparece un bloque comparador.

4. Diseño del Hardware asociado a la Sincronización Tiempo-Frecuencia

4.1 Módulo de Sincronización en frecuencia

El diseño del módulo que desempeña el algoritmo de sincronización en frecuencia, ha sido realizado utilizando bloques de las librerías *Xilinx Blockset y Xilinx Reference Blockset de Xilinx System Generator for DSP®* [4].Estos bloques fueros descritos en el capítulo 4. El algoritmo que se ha descrito en hardware se corresponde con la adaptación del algoritmo propuesto en [1], que anteriormente explicamos en el apartado (2.). Este módulo de estimación del desplazamiento frecuencial está dividido en cuatro bloques principales, los cuales se muestran en la figura (6.3).

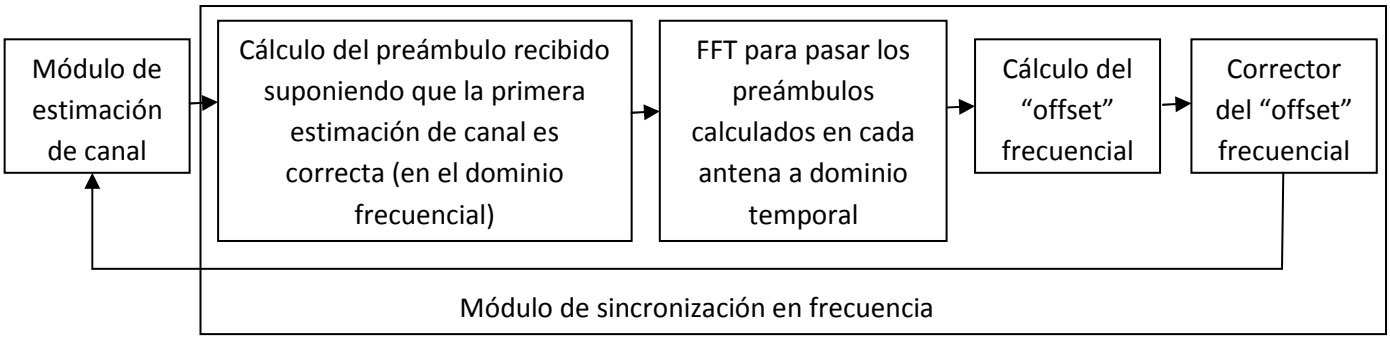


Figura 6.3: Módulo de sincronización en frecuencia

Antes de comenzar con la descripción del módulo de sincronización en frecuencia, es preciso tener en cuenta, previamente, una serie de consideraciones generales:

- Para comprobar el correcto funcionamiento del hardware diseñado, es preciso generar las señales de entrada a dicho hardware. Las señales de entrada deberán de corresponderse con el preámbulo recibido, el cual habrá sufrido un desplazamiento en frecuencia. Para ello, se ha simulado en Matlab® el paso del preámbulo a través del canal, con dicho desplazamiento en frecuencia. La función de Matlab® que simula el paso del preámbulo por el canal es "*MLTF_offset.m*", la cual realiza una llamada a otra función de Matlab® ("*chan_SUI.m*") que simula uno de los 4 canales SISO del canal 2x2 MIMO. Esta función, nos devolverán la

parte real y la parte imaginaria de los preámbulos recibidos en cada antena receptora en los vectores "*En_re_Pre_out_ant1_time, En_im_Pre_out_ant1_time, En_re_Pre_out_ant2_time y En_im_Pre_out_ant2_time*". Seremos capaces de importar los datos guardados en esos vectores generados por la función de Matlab® mediante el uso de bloques "***From Workspace***".

- Las dos funciones de Matlab®, anteriormente comentadas, las podemos encontrar en el Anexo A.
- El modelo de canal simulado en Matlab® es el mismo que empleamos en el capítulo de estimación de canal
- Para definir los puertos de entrada al hardware diseñado, se han utilizado bloques "***Gateway In***" configurados para trabajar con datos tipo "*signed*" de 24 bits con el punto binario en el bit 14. Esto se ha realizado así, para poder representar de una manera precisa la información, de lo contrario la calidad y precisión de la estimación del desplazamiento frecuencial no sería la misma. Este número de bits se trata de mantener a lo largo del diseño.
- Al igual que en módulo de estimación de canal, el sincronismo de todos los elementos síncronos del módulo de sincronización en frecuencia está controlado por un contador común. Este contador común, será utilizado para generar las señales de control que gestionan distintos elementos del diseño.

➔ *Visión general del modelo Simulink®*

Debido a la complejidad del módulo de sincronización en frecuencia, en este punto, mostraremos la visión general del modelo Simulink® particularizado para cada uno de los bloques principales mostrados en la figura (6.4). En la figura (6.4) se ilustra la apariencia que tienen los cuatro bloques principales del diseño, los cuales hemos resaltado en recuadros y numerado. En apartados sucesivos, resaltaremos los elementos más importantes de cada uno de estos bloques.

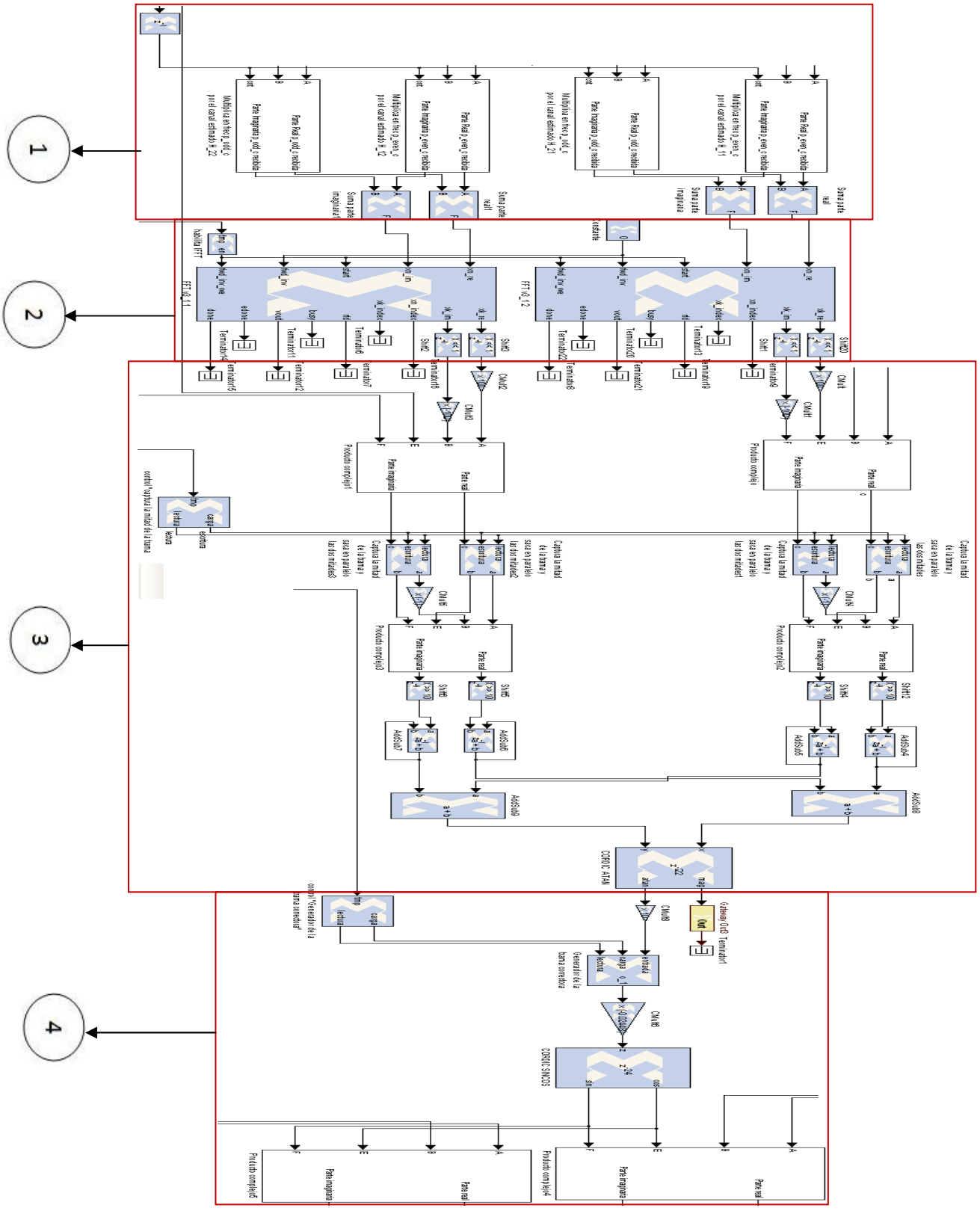


Figura 6.4: Módulo de sincronización en frecuencia

4.1.1 Bloque 1:"Calculo del preámbulo recibido suponiendo que la primera estimación de canal es correcta (en el dominio frecuencial)"

En la figura (6.5), mostramos el aspecto general que tiene este bloque del diseño. Sobre ella, resaltaremos las partes más importantes mediante recuadros y las numeraremos para describirlas más adelante.

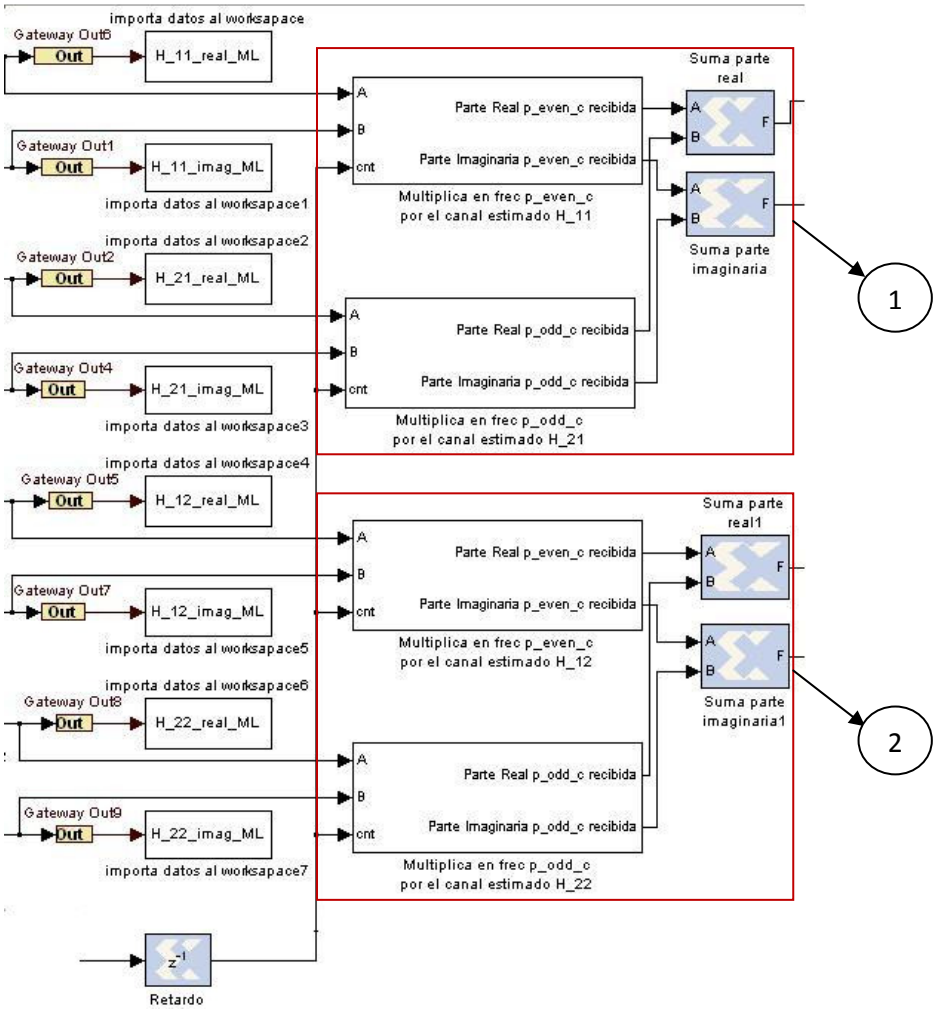


Figura 6.5: Bloque 1:"Calculo del preámbulo recibido suponiendo que la primera estimación de canal es correcta (en el dominio frecuencial)"

1) Cálculo del preámbulo recibido en la antena 1 suponiendo que la primera estimación de canal es correcta ($\underline{Y}(1) = \underline{H}_{11}\underline{x}_{P_{EVEN}} + \underline{H}_{21}\underline{x}_{P_{ODD}}$).

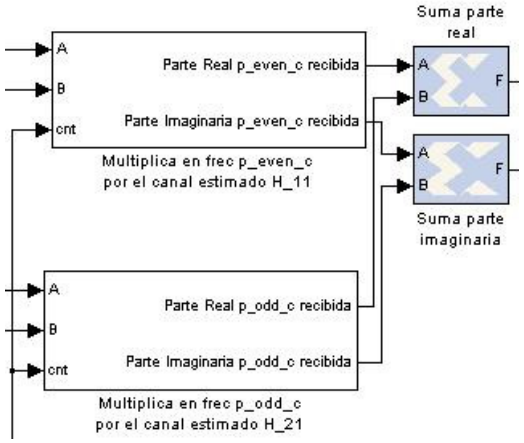


Figura 6.6: Etapa 1:"Calculo del preámbulo que se recibiría en la antena 1"

Esta etapa del diseño (figura (6.6)), calcula el preámbulo que se recibiría en la antena 1 si la estimación de canal fuera correcta. Esta operación está descrita por la expresión (6.6), la cual recordamos a continuación:

$$\underline{Y}(1) = \underline{H}_{11}\underline{x}_{P_{EVEN}} + \underline{H}_{21}\underline{x}_{P_{ODD}} \qquad (6.16)$$

Las señales de entrada a esta etapa del diseño, se corresponden con los canales estimados H11 y H21, los cuales serán necesarios para el cálculo de la expresión (6.16).

Esta etapa está compuesta por dos subsistemas y dos bloques sumadores. Cada uno de los subsistemas, se encarga de calcular los sumandos que intervienen en la expresión (6.16). Concretamente, el subsistema "*Multiplica p_even_c por el canal estimado H_11*", se encarga de realizar el producto complejo $\underline{H}_{11}\underline{x}_{P_{EVEN}}$ que no es más que multiplicar la trama de 256 muestras del canal H11 estimado por la trama de 256 muestras del preámbulo con información en las subportadoras pares. Por otra parte, el subsistema "*Multiplica p_odd_c por el canal estimado H_21*", se encarga de realizar el producto complejo $\underline{H}_{21}\underline{x}_{P_{ODD}}$

que no es más que multiplicar la trama de 256 muestras del canal H11 estimado por la trama de 256 muestras del preámbulo con información en las subportadoras impares. En la figura (6.7), se muestra como es el interior de cada uno de estos subsistemas.

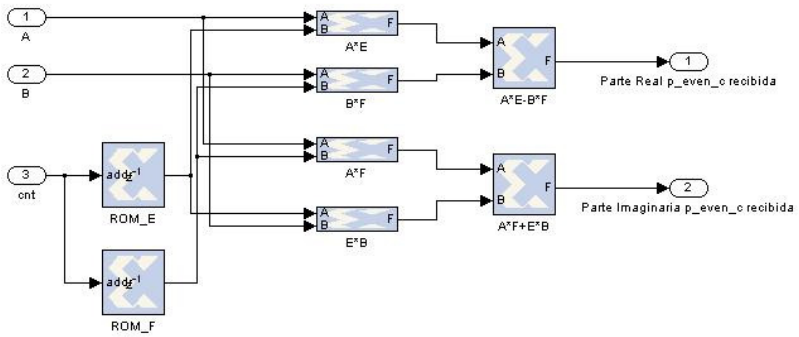


Figura 6.7: Subsistema "Multiplica en frec p_even_c por el canal estimado H_11"

Realizar un producto complejo, implica realizar las siguientes operaciones:

$$producto\ complejo = [A + Bj][E + Fj]$$
$$= (AE - BF) + (AF + EB)j \quad (6.17)$$

Por lo tanto, un producto complejo implica realizar 4 multiplicaciones, una resta y una suma. Así, los subsistemas implicados en esta etapa del diseño están constituidos por 4 multiplicadores, un restador y un sumador, conectados de tal manera, que a la salida de cada subsistema se tiene el producto complejo deseado. En la figura (6.7), se puede observar el conexionado que acabamos de comentar.

El preámbulo conocido, está guardado en las memorias ROM que aparecen dentro de cada subsistema, tal y como se observa en la figura (6.7). Concretamente, en el subsistema "*Multiplica p_even_c por el canal estimado H_11*", la memoria *ROM_E* guarda la parte real del preámbulo P_{EVEN} y la memoria *ROM_F* guarda la parte imaginaria del preámbulo P_{EVEN}. Por otra parte, en el subsistema "*Multiplica p_odd_c por el canal estimado H_21*", la memoria *ROM_E* guarda la parte real del preámbulo

P_{ODD} y la memoria *ROM_F* guarda la parte imaginaria del preámbulo P_{ODD}. Cada dirección de memoria hará referencia a cada subportadora del preámbulo.

Las memorias anteriormente descritas, son leídas secuencialmente de manera que se va leyendo de la dirección de memoria 0 a la 255, es decir, se va leyendo una subportadora tras otra del preámbulo. Las direcciones de memoria son generadas por el mismo bloque contador de 0 a 255, que se utilizaba para leer las memorias del módulo de estimación de canal (ver apartado (3.5) del capítulo 5). Únicamente, se ha retardado un ciclo de reloj la salida de datos del contador, mediante un bloque "***Delay***" (ver figura (6.5)), para sincronizar la lectura de las memorias con el instante de llegada de los datos provenientes del módulo de estimación de canal.

En las figuras (6.8), (6.9), (6.10) y (6.11), se muestran los cuadros de configuración de los elementos utilizados en cada subsistema. Los bloques multiplicadores, sumadores y restadores están modelados mediante bloques "***Black Box***", los cuales tendrán un código VHDL asociado que podemos encontrar en el anexo B.

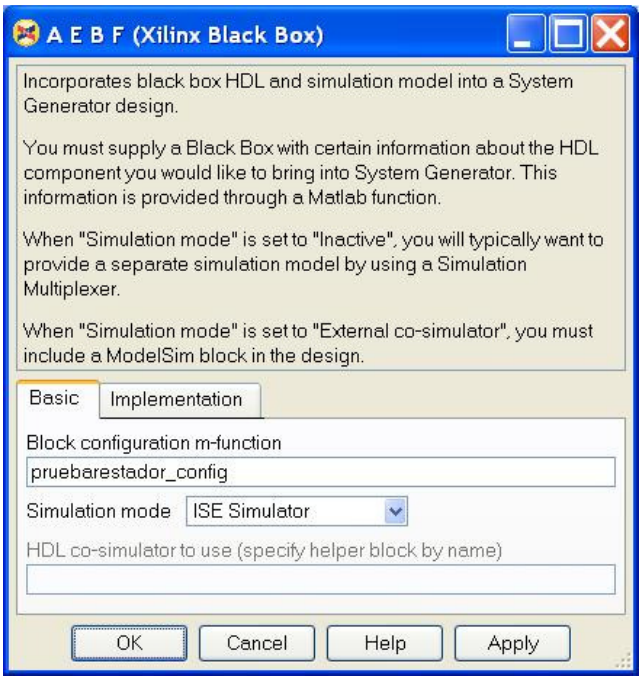


*Figura 6.8: Cuadro de configuración del bloque "Black Box" utilizado para modelar el restador A*E-B*F*

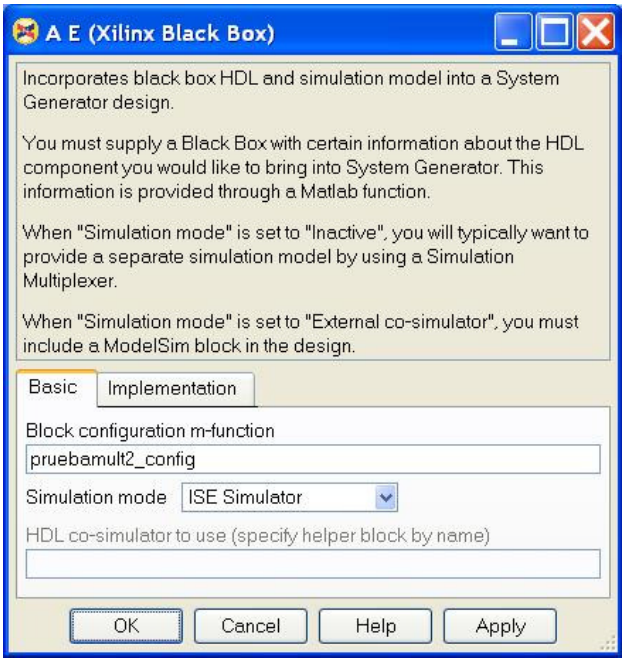


*Figura 6.9: Cuadro de configuración del bloque "Black Box" utilizado para modelar el multiplicador A*E*

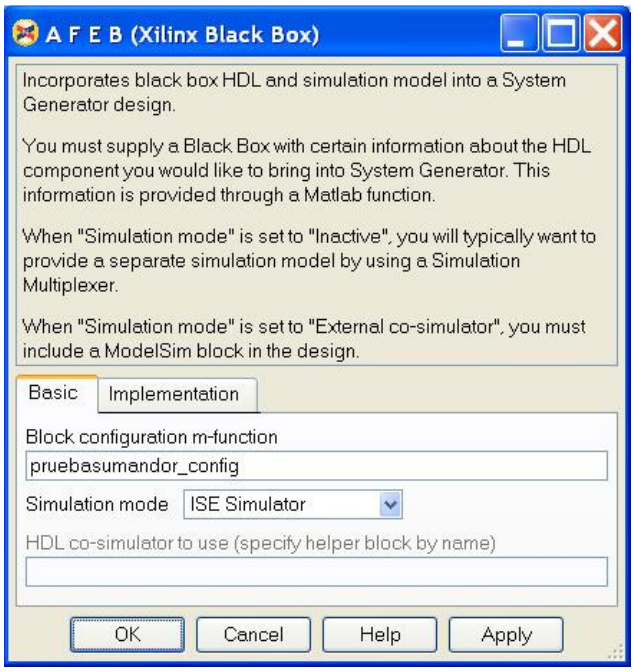


*Figura 6.10: Cuadro de configuración del bloque "Black Box" utilizado para modelar el sumador A*F+E*B*

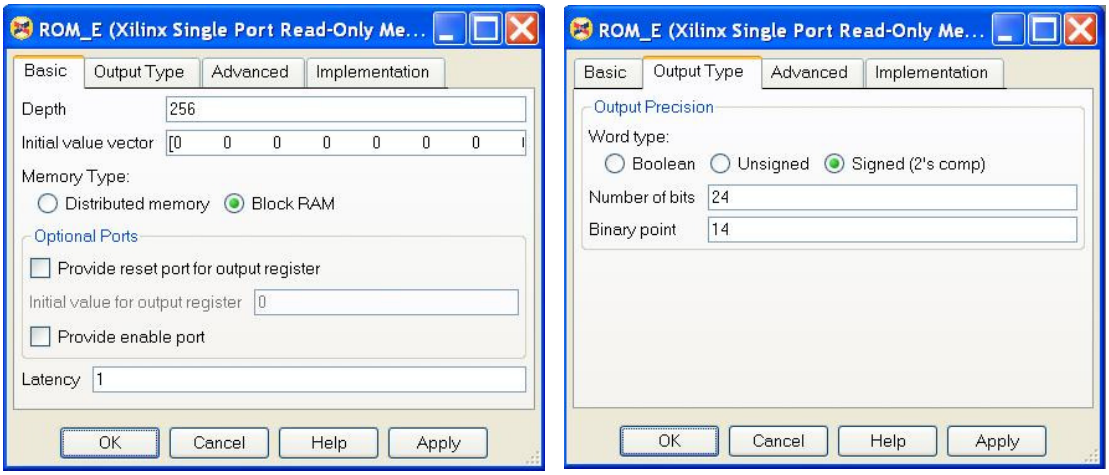


Figura 6.11: Cuadro de configuración de los bloques ROM (ROM_E)

Por último, las salidas de cada subsistema se conectan a dos sumadores para terminar de completar la suma de la expresión (6.16). Un sumador suma las partes reales de los productos complejos previamente calculados, y el otro suma las partes imaginarias. En cuadro de configuración de estos dos últimos sumadores se ilustra en la figura (6.12).

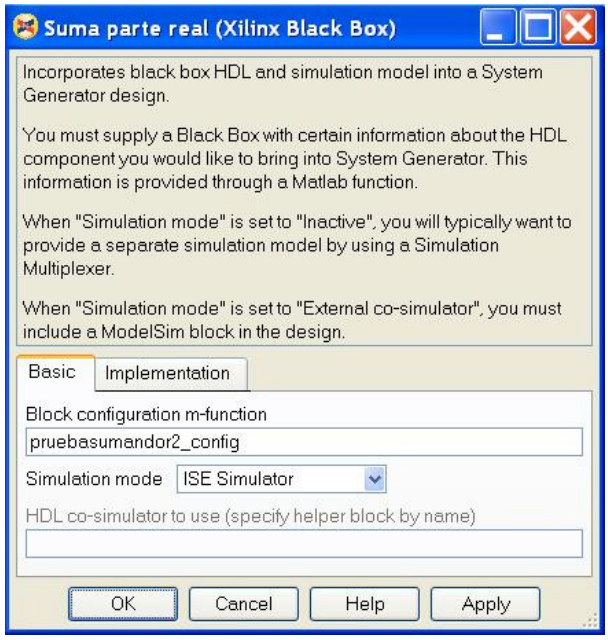


Figura 6.12: Cuadro de configuración del bloque "Black Box" utilizado modelar los sumadores de salida

2) Cálculo del preámbulo recibido en la antena 2 suponiendo que la primera estimación de canal es correcta ($\underline{Y}(1) = \underline{H}_{11}\underline{x}_{P_{EVEN}} + \underline{H}_{21}\underline{x}_{P_{ODD}}$).

Esta parte del diseño esta realizada de la misma manera que la descrita en el apartado anterior, pero referida a la antena receptora 2. Por tanto, no es necesaria su descripción.

4.1.2 Bloque 2: "FFT para pasar los preámbulos calculados en cada antena a dominio temporal"

En la figura (6.12), mostramos el aspecto general que tiene este bloque del diseño. Sobre ella, resaltaremos las partes más importantes mediante recuadros y las numeraremos para describirlas más adelante.

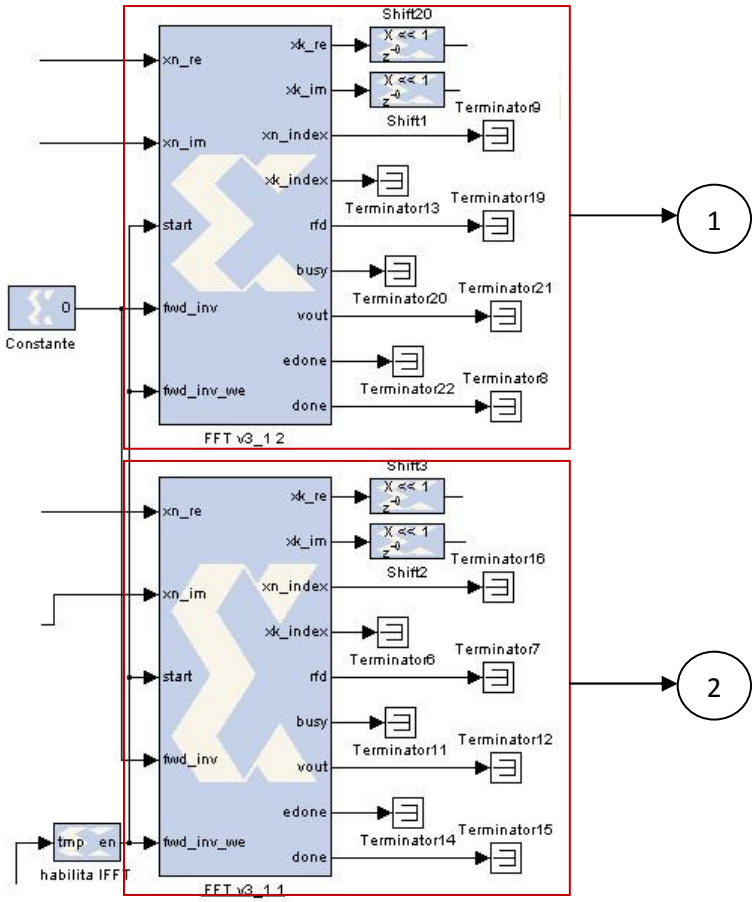


Figura 6.13: Bloque 2: "FFT para pasar los preámbulos calculados en cada antena a dominio temporal"

1) Transformación del preámbulo calculado para la antena 1 en el bloque anterior al dominio del tiempo ($\underline{y}_t(1)'$).

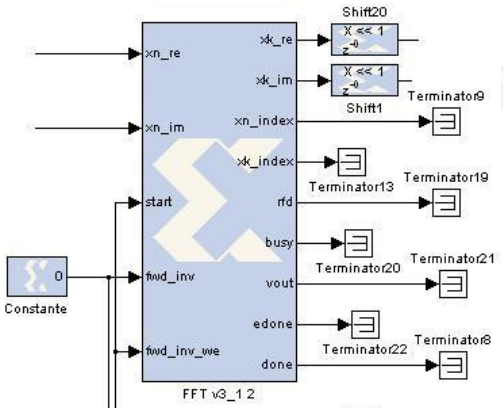


Figura 6.14: Etapa 1:"Transformación del preámbulo calculado en la etapa anterior al dominio del tiempo"

Esta etapa del diseño (figura (6.14)), realiza la IFFT del preámbulo calculado en el bloque anterior, para obtener el preámbulo en el dominio del tiempo. Esta operación está descrita por la expresión (6.8), la cual recordamos a continuación:

$$\underline{Y}(q) \rightarrow IDFT\ 256\ puntos \rightarrow \underline{y}_t(q)' \qquad (6.18)$$

Para ello, se emplea un bloque "***FFT v3_1***" configurado para realizar una IFFT de 256 puntos. Es de destacar en la figura (6.13) como están conectadas las entradas del bloque "***FFT v3_1***". Los aspectos que debemos destacar del bloque "***FFT v3_1***" son:

➔ La entrada "*Start*" está conectada a una entrada de habilitación que determina cuando el bloque "***FFT v3_1***" debe comenzar a capturar la trama de entrada para comenzar a realizar la IFFT. Esta entrada de habilitación está generada por el bloque "*habilita IFFT*". Este bloque tiene como entrada de datos la salida del contador común, anteriormente descrito, de manera que cuando el contador alcanza un determinado valor, el bloque genera la señal de habilitación que determina el comienzo de la IFFT.

El bloque "*habilita IFFT*" es un bloque "***Black Box***", el cual tiene asociado un código VHDL que podemos encontrar en el anexo B. El cuadro de configuración de este bloque lo podemos encontrar en la figura (6.15).

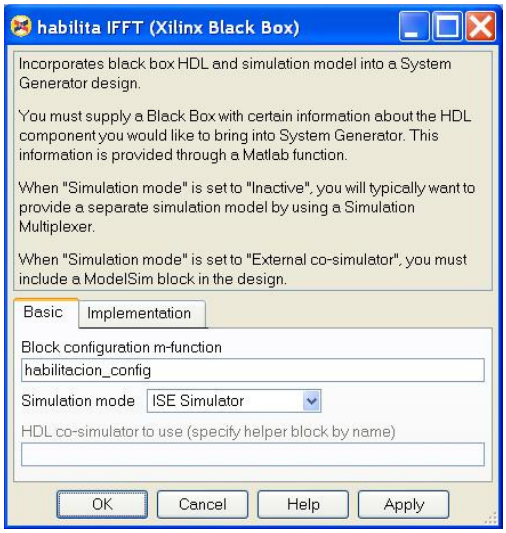


Figura 6.15: Cuadro de configuración del bloque "Black Box" utilizado modelar la señal de habilitación

➔ El bloque "***FFT v3_1***" tiene una entrada para la parte real y otra para la parte imaginaria *(xn_re y xn_im).* Hay que tener en cuenta que, el bloque "***FFT v3_1***" admite datos de entrada tipo "*signed*" de 8, 12, 16, 20 o 24 bits con el punto binario situado de tal manera que solo tenemos un bit para representar la parte entera del dato. Sin embargo, en este caso, no es necesario utilizar bloques adicionales para adaptar los datos de entrada, puesto que los datos de salida de los sumadores de la etapa anterior ya están adaptados al efecto comentado. Simplemente, habrá que adaptar los datos de salida y poner el punto binario en su sitio para obtener datos tipo "*signed*" de 24 bits y punto binario en el bit 14. Así, hacemos pasar los datos de salida por bloques "***Shift***" (ver figura (6.14)) configurados como se puede observar en la figura (6.16).

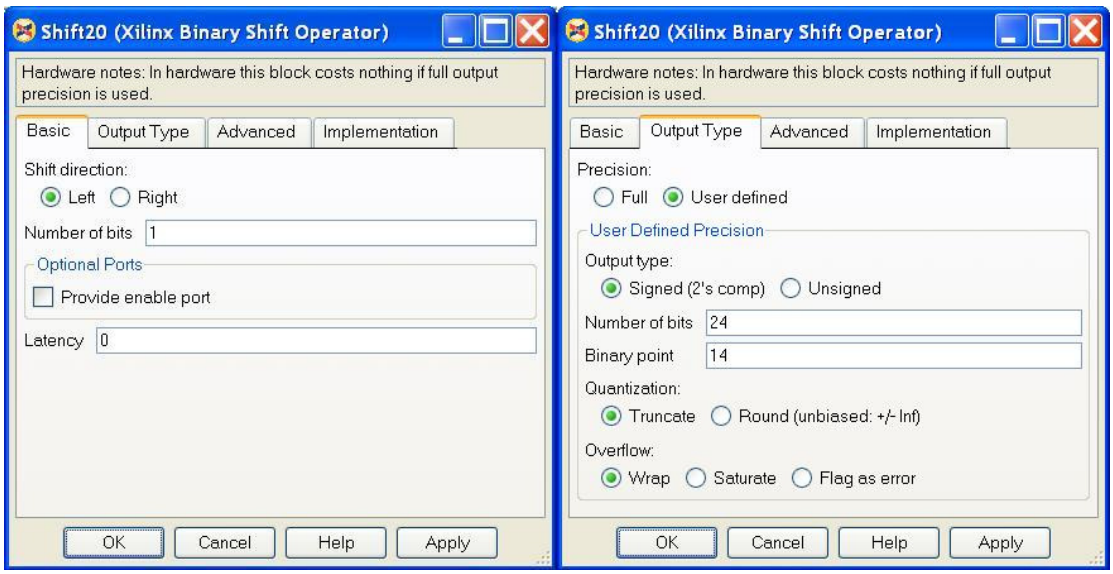


Figura 6.16: Cuadro de configuración de los bloques "Shift" utilizados a la salida

➔ La entrada "*fw_inv*", indica si el bloque está configurado para realizar una FFT o una IFFT. En este caso, como se quiere hacer una IFFT, utilizamos como dato de entrada una constante de valor 0 tipo booleano.

➔ La entrada "*fw_inv_we*" se utiliza para cargar el valor de la entrada "*fw_inv*", que indicará si se realiza una transmformada inversa o no. Cuando esta activa, se carga el valor de "*fw_inv*". Como podemos observar en la figura (6.12), la entrada "*fw_inv_we*" está conectada al bloque "*habilita IFFT*" para que se cargue el tipo de transformada en el mismo instante que se habilita el bloque para que comience a realizar la IFFT.

En la figura (6.17), podemos observar cómo está configurado uno de los bloques FFT v3_1 de esta etapa del diseño.

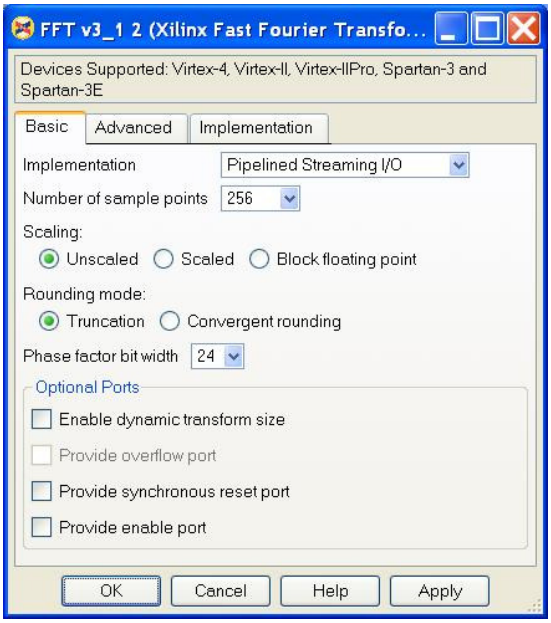


Figura 6.17: Cuadro de configuración de los bloques "FFT v3_1"

Esta etapa nos permitirá tener a su salida el preámbulo calculado previamente en el dominio del tiempo, para que pueda ser manipulado por el algoritmo de sincronización frecuencial que estamos tratando, el cual trabaja en el dominio del tiempo.

2) Transformación del preámbulo calculado para la antena 2 en el bloque anterior al dominio del tiempo ($\underline{y}_t(2)'$).

   Esta parte del diseño esta realizada de la misma manera que la descrita en el apartado anterior, pero referida a la antena receptora 2. Por tanto, no es necesaria su descripción.

4.1.3 Bloque 3: "Cálculo del "*offset*" frecuencial"

En la figura (6.18), mostramos el aspecto general que tiene este bloque del diseño. Sobre ella, resaltaremos las partes más importantes mediante recuadros y las numeraremos para describirlas más adelante.

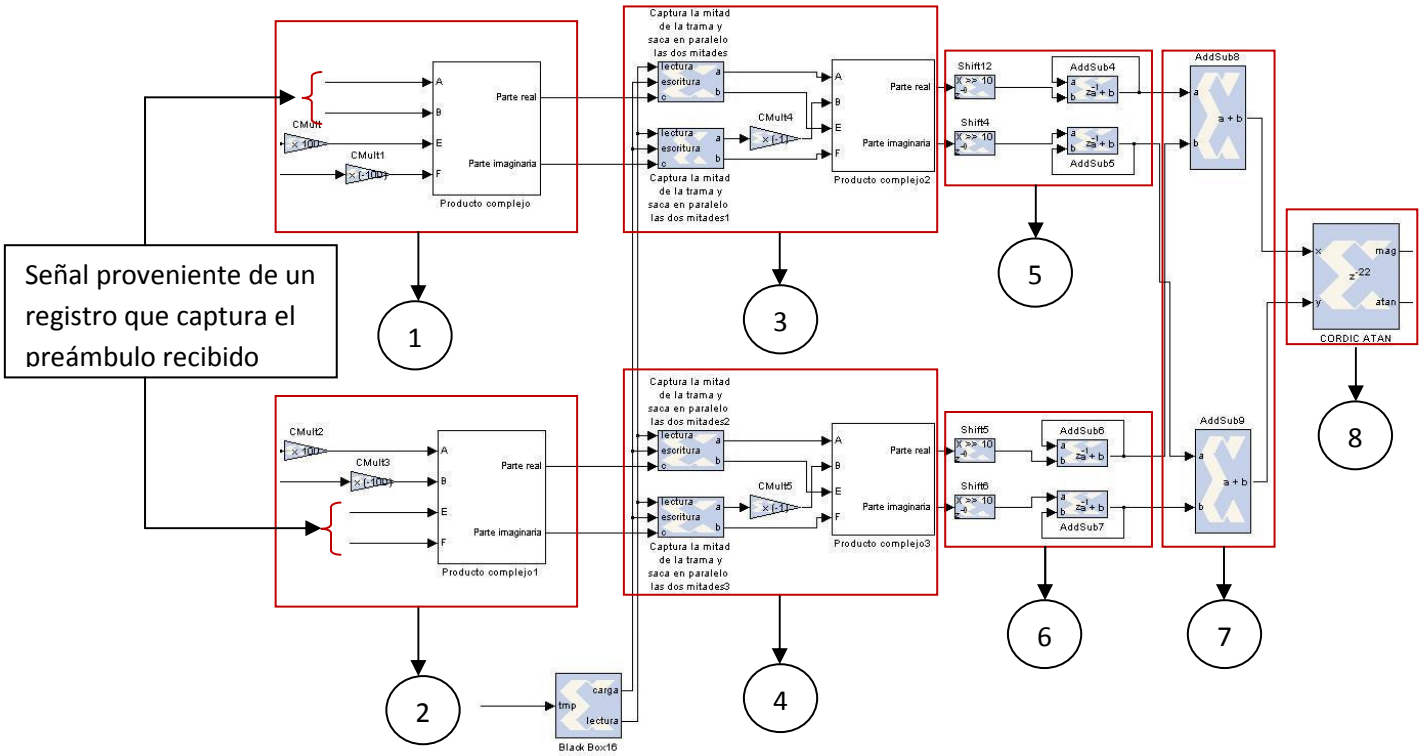


Figura 6.18: Bloque 3: "Cálculo del "offset" frecuencial"

1) Multiplica el preámbulo real recibido en la antena 1 por el conjugado del preámbulo calculado en la etapa anterior, el cual es el preámbulo que se habría recibido suponiendo que la primera estimación de canal fuese correcta.

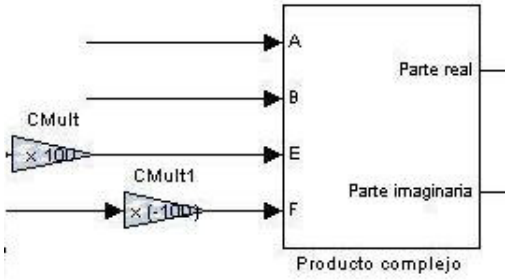
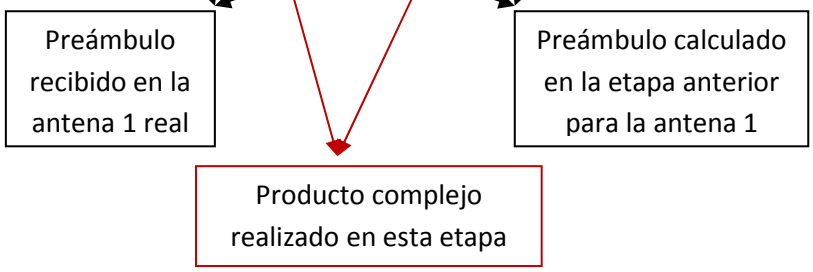


Figura 6.19: Etapa 1: "Producto complejo entre el preámbulo calculado y recibido"

Esta etapa del diseño (figura (6.19)), multiplica el conjugado del preámbulo calculado en la etapa anterior para la antena 1 por el preámbulo que realmente se recibe. Con esta etapa lo que se está calculando son los productos complejos que hay en la expresión (6.10), la cual recordamos a continuación particularizada para la antena 1.

$$\hat{\epsilon}_{frac_1} = angle\left(\sum_{t=0}^{127}\left(\underline{y}_t(1,1)\,\underline{y}_t(1,1)'^*\right)^*\left(\underline{y}_t(1,2)\,\underline{y}_t(1,2)'^*\right)\right) \qquad (6.19)$$



En la etapa anterior, se ha calculado el preámbulo que se recibiría si la estimación de canal fuese correcta, es decir, indirectamente, en la etapa anterior, se ha simulado el paso del preámbulo original por el canal de la primera estimación. Este preámbulo sufre la atenuación del canal estimado, por lo que será necesario compensarlo. Así, las salidas de los bloques "***FFT v3_1***" de la etapa anterior (ver figura (6.18)), la cuales son dos de las entradas a esta etapa, serán multiplicadas por 100 mediante el empleo de bloques "***CMult***" configurados según la figura (6.20). Notar como el bloque "***CMult***" de la parte imaginaria tiene como valor multiplicativo -100, ya que lo que se quiere es hacer el conjugado.

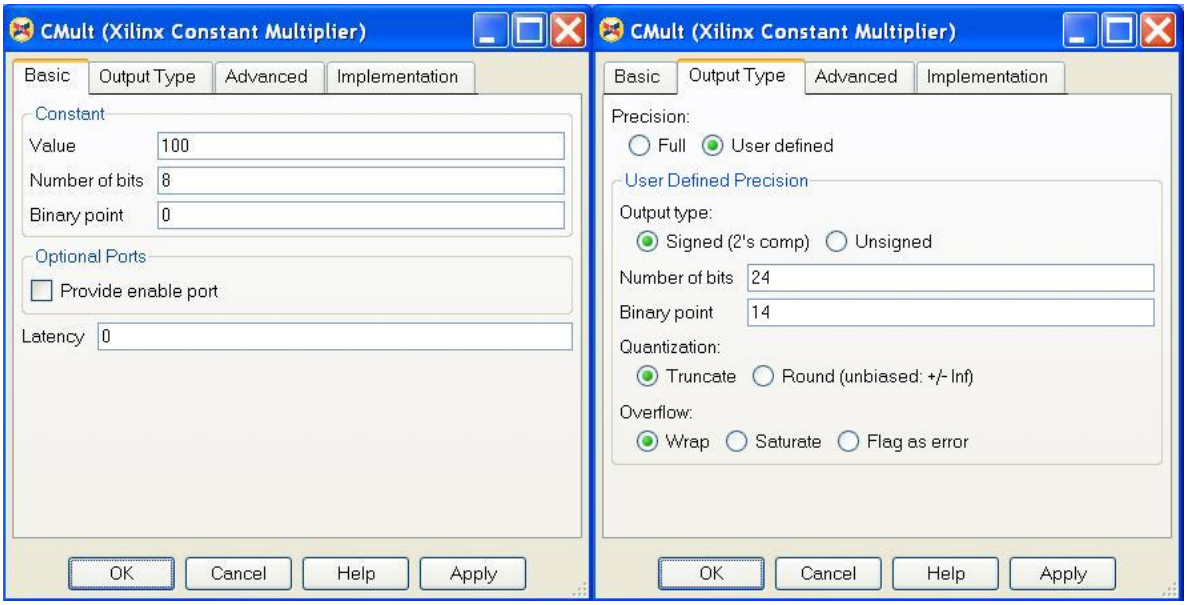


Figura 6.20: Cuadro de configuración de los bloques "CMult" empleados en la etapa 1

Por otra parte, las otras dos entradas a esta etapa son la parte real e imaginaria del preámbulo que realmente se recibe. Este preámbulo recibido se almacena en un registro en el momento de su recepción. De este registro se leerá el preámbulo recibido en el momento que se desee. Así, se sincroniza la lectura de este registro con la llegada de los datos del preámbulo calculado en la etapa anterior, de manera que los productos complejos de la expresión (6.19) se realicen correctamente. Esto se ha diseñado así, para evitar tener que utilizar bloques "***Delay***" para alinear el preámbulo original recibido con el calculado en la etapa anterior. Utilizar bloques "***Delay***" que introdujesen una gran retardo, suponía el empleo de un número excesivo de registros, disparando así el área del diseño.

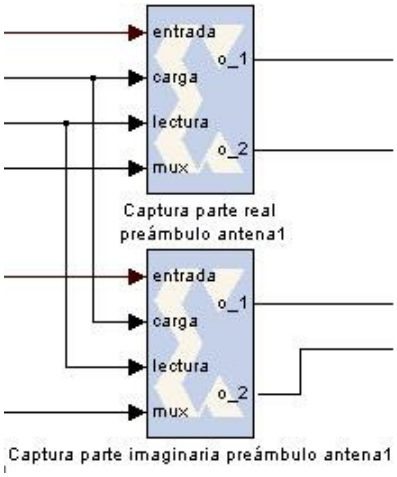


Figura 6.21: Registro que almacena el preámbulo recibido en la antena 1

Se ha descrito un registro para almacenar la parte real del preámbulo recibido y otro registro para almacenar la parte imaginaria, tal y como se puede ver en la figura (6.21). Estos registros se pueden identificar en la figura (6.21), bajo los nombres "*Captura parte real preámbulo antena1*" y "*Captura parte imaginaria preámbulo antena1*". Cada registro posee las siguientes entradas: "*entrada*", "*carga*", "*lectura*" y "mux"; y las siguientes salidas: "*o_1*" y "*o_2*". Cuando "*carga*" esta activa se capturan los datos a la entrada y cuando "*lectura*" esta activa se leen los datos almacenados en el registro. Por otra parte, "*mux*" sirve para determinar por cuál de las dos salidas se sacarán los datos leídos. Estos registros están modelados mediante bloques "***Black Box***", los cuales tienen un código VHDL asociado que podemos encontrar en el anexo B. El cuadro de configuración de uno de estos bloques, se ilustra en la figura (6.22).

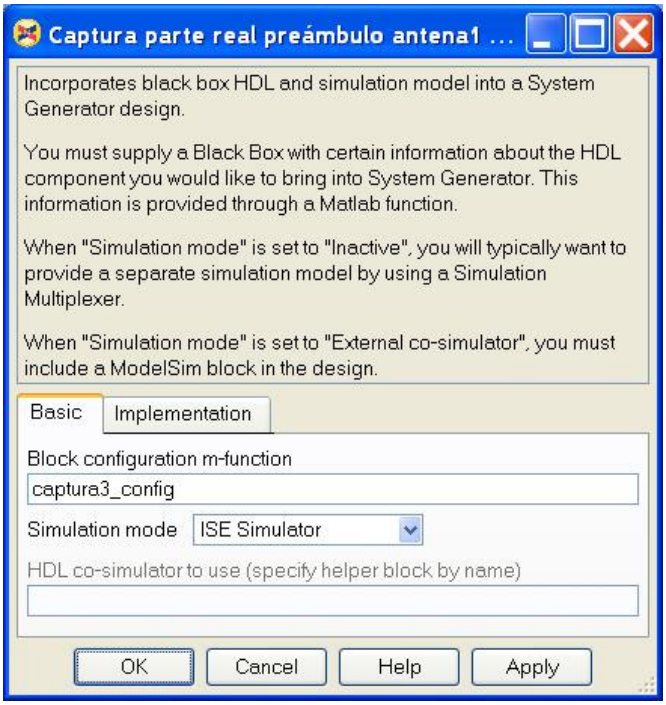


Figura 6.22: Cuadro de configuración de los bloques "Box Black" empleados para el modelado de los registros

Las entradas "*carga*","*lectura*" y"*mux*" son entradas que sirven para controlar el funcionamiento de los registros. Estas entradas están conectadas a un bloque "***Black Box***", el cual genera las señales de control que necesarias para el correcto funcionamiento de los registros anteriores. Este bloque, a su vez, está controlado por el contador común que comentamos en el capítulo 5. Este bloque "***Black Box***", identificado bajo el nombre "*control registros*", tiene un código VHDL asociado, el cual podemos consultar en el anexo B. En la figura (6.23), se muestra la apariencia de este bloque en el modelo Simulink® y su cuadro de configuración.

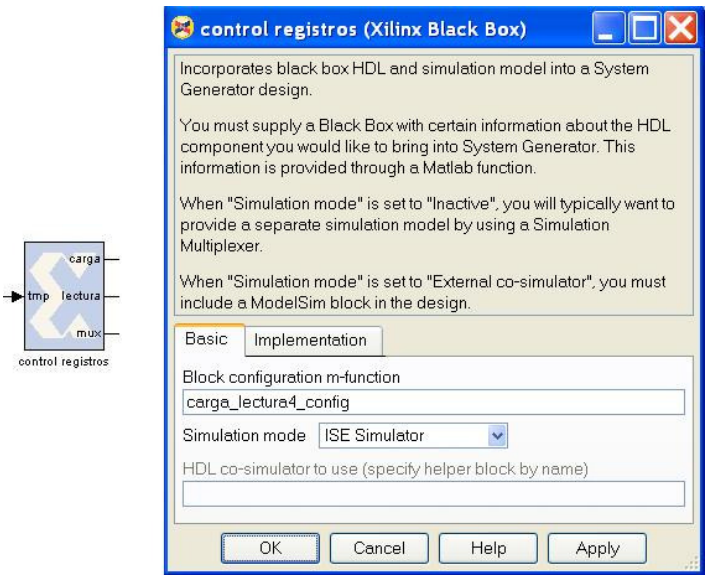


Figura 6.23: Apariencia y cuadro de configuración del bloque "Box Black" empleado para el modelado del bloque que controla los registros

Por otra parte, la entrada de datos a esta etapa se realiza en serie. Aprovechando la entrada serie de los datos, se va haciendo el producto complejo en cada subportadora de manera secuencial. Así, como ya vimos en apartados anteriores, el un producto complejo implica la realización de 4 multiplicaciones, una resta y una suma. Así, en el subsistema bajo el nombre "*Producto complejo*" (ver figura (6.19)), aparecen 4 multiplicadores, un restador y un sumador conectados de tal manera que se realiza un producto complejo. Esto se ilustra en la figura (6.24).

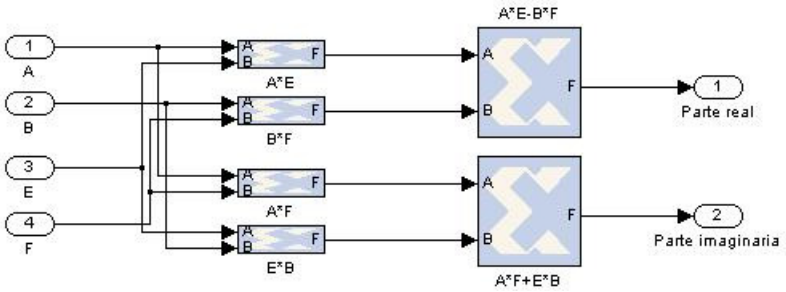


Figura 6.24: Interior del subsistema "Producto complejo"

En las figuras (6.25), (6.26) y (6.27), se muestran los cuadros de configuración de los elementos utilizados en el subsistema "*Producto complejo*". Los bloques multiplicadores, sumadores y restadores están modelados mediante bloques "***Black Box***", los cuales tendrán un código VHDL asociado que podemos encontrar en el anexo B.

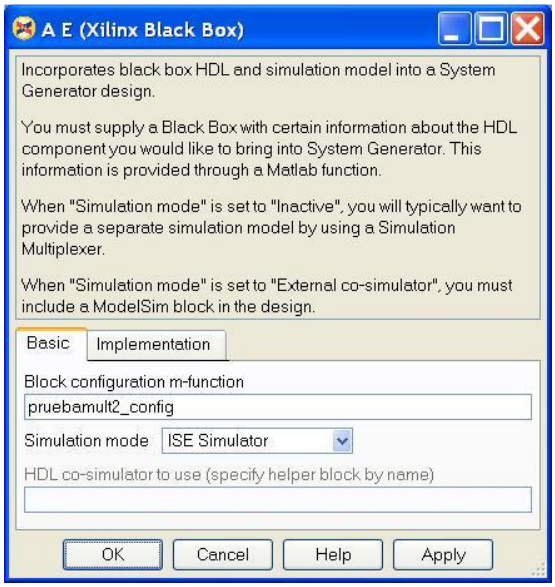


*Figura 6.25: Cuadro de configuración del bloque "Black Box" utilizado para implementar el multiplicador A*E*

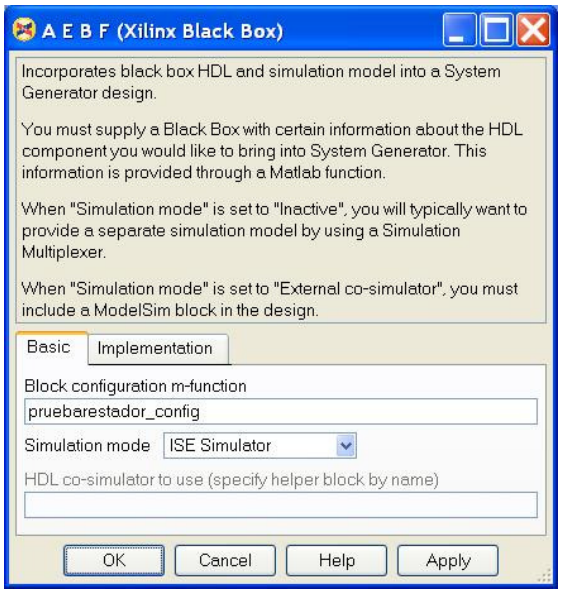


*Figura 6.26: Cuadro de configuración del bloque "Black Box" utilizado para implementar el restador A*E-B*F*

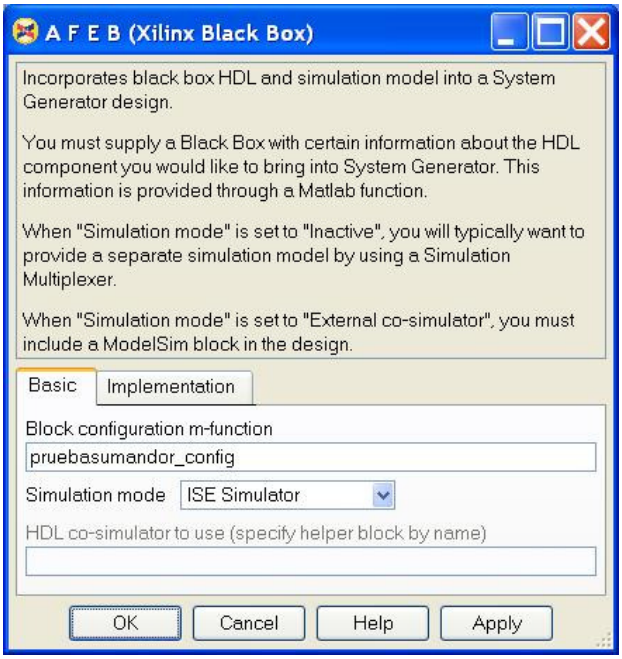


*Figura 6.27: Cuadro de configuración del bloque "Black Box" utilizado para modelar el sumador A*F-E*B*

2) Multiplica el preámbulo real recibido en la antena 2 por el conjugado del preámbulo calculado en la etapa anterior, el cual es el preámbulo que se habría recibido suponiendo que la primera estimación de canal fuese correcta.

   Esta parte del diseño esta realizada de la misma manera que la descrita en el apartado anterior, pero referida a la antena receptora 2. Por tanto, no es necesaria su descripción.

3) Pone en paralelo las dos mitades de la trama recibida y realiza el producto complejo del conjugado de la primera mitad de la trama recibida por la segunda mitad de la trama recibida (antena 1).

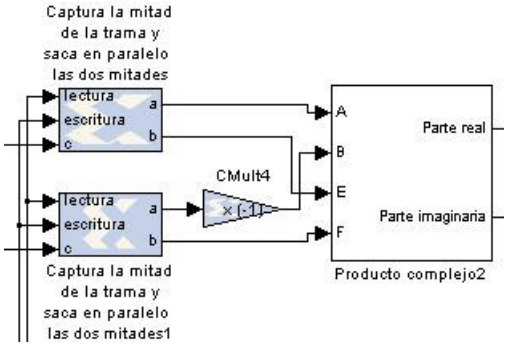


Figura 6.28: Etapa 3: "Pone en paralelo las dos mitades de la trama recibida y realiza el producto complejo del conjugado de la primera mitad de la trama recibida por la segunda mitad de la trama recibida (antena 1)"

Esta etapa del diseño (figura (6.28)), realiza el producto complejo que aparece en la expresión (6.10), la cual recordamos a continuación particularizada para la antena 1:

$$\hat{\epsilon}_{frac_1} = angle\left(\sum_{t=0}^{127}\left(\underline{y}_t(1,1)\,\underline{y}_t(1,1)'^{*}\right)^{*}\left(\underline{y}_t(1,2)\,\underline{y}_t(1,2)'^{*}\right)\right) \qquad (6.20)$$

Primera mitad

Operación realizada en esta etapa

Segunda mitad

Para ello, esta etapa desdobla los datos generados en la etapa anterior en dos líneas y realiza el producto complejo que marca la expresión (6.20). El desdoble se lleva a cabo, en los bloques identificados como "*captura la mitad de la trama y saca en paralelo las dos mitades*", tal y como podemos observar en la figura (6.28). Se ha diseñado dos bloques de este tipo, uno para la parte imaginaria y otro para la parte real. Cada uno de estos bloques posee como entradas: "*lectura*", "*escritura*" y "*c*"; y como salidas: "*a*" y "*b*". El funcionamiento de uno de estos bloques es el siguiente:

- Cuando "*escritura*" esta activa el bloque captura en un registro los datos de entrada en "*c*"(esto sucede para las 128 primeras muestras provenientes de la etapa anterior)

- Cuando "*lectura*" esta activa se leen los datos capturados, los cuales se sacan por la salida "*a*". En este momento, al mismo tiempo, se sacan por "*b*" los datos que están llegando a la entrada por "*c*". De esta manera, se consigue desdoblar la trama de entrada en dos líneas y ponerlas en paralelo en el mismo instante de tiempo.

Los bloques identificados como "*captura la mitad de la trama y saca en paralelo las dos mitades*" son bloques "***Black Box***", los cuales tienen asociados códigos VHDL que podemos encontrar en el anexo B. En la figura (6.29), podemos ver como es su cuadro de configuración.

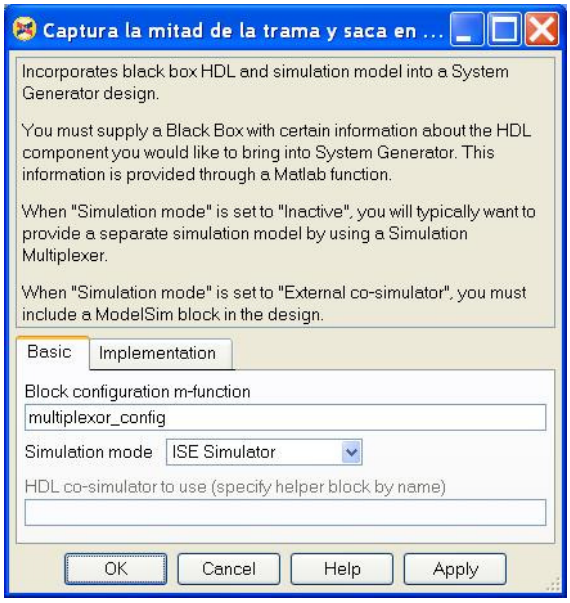


Figura 6.29: Cuadro de configuración del bloque "Black Box" utilizado para modelar el bloque que desdobla las líneas y las pone en paralelo

Para ilustrar el funcionamiento de uno de los bloques "*captura la mitad de la trama y saca en paralelo las dos mitades*", se ilustra una captura del visor "***Wavescope***" en la figura (6.30).

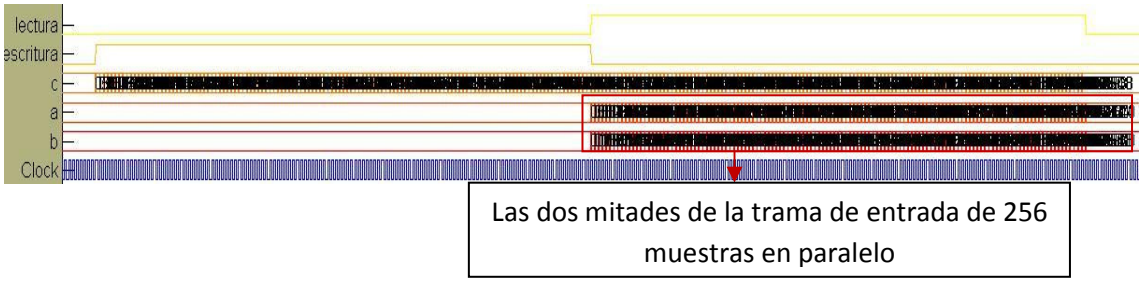


Figura 6.30: Ilustración del funcionamiento del bloque "captura la mitad de la trama y saca en paralelo las dos mitades"

Los bloques anteriormente descritos, son controlados por un bloque "***Black Box***" identificado bajo el nombre "*control "captura la mitad de la trama y saca en paralelo las dos mitades"*". Este bloque está controlado, a su vez, por el contador común comentado en el capítulo 5, de manera que cuando este lleva una determinada cuenta, este bloque de control, genera a su salida las señales que gestionan los bloques "*captura la mitad de la trama y saca en paralelo las dos mitades*". Este bloque de control, tiene asociado un código VHDL que podemos encontrar en el anexo B. En la figura (6.31), se muestra la apariencia de este bloque en el modelo Simulink® junto con su cuadro de configuración.

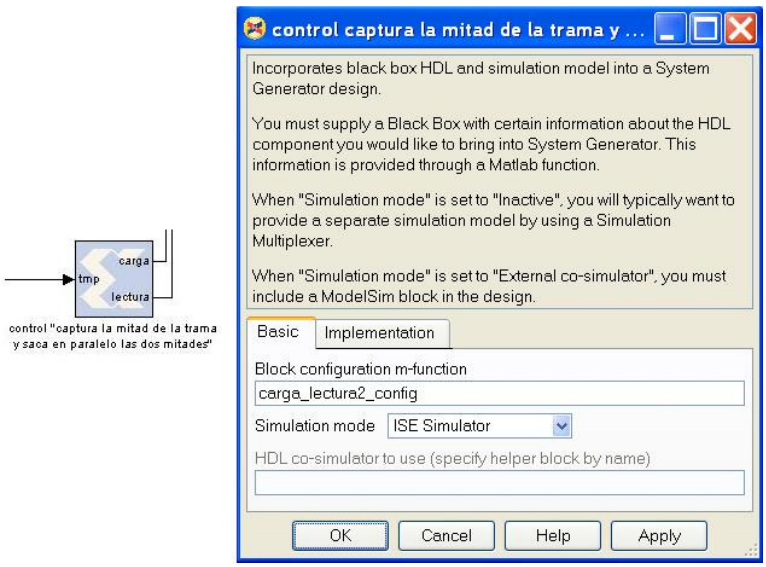


Figura 6.31: Apariencia y cuadro de configuración del bloque "Box Black" empleado para el modelado del bloque "Control "captura la mitad de la trama y saca en paralelo las dos mitades""

Una vez tenemos los datos generados en la etapa anterior con las dos mitades de la trama recibida en paralelo, podemos empezar a realizar el producto complejo definido en la expresión (6.20). Antes de realizar el producto complejo, debemos realizar el conjugado de la primera mitad de la trama. Para ello, multiplicaremos por -1 los datos de la salida "a" del bloque "*captura la mitad de la trama y saca en paralelo las dos mitades1*". Esto se realiza mediante un bloque "***CMult***" configurado según la figura (6.32).

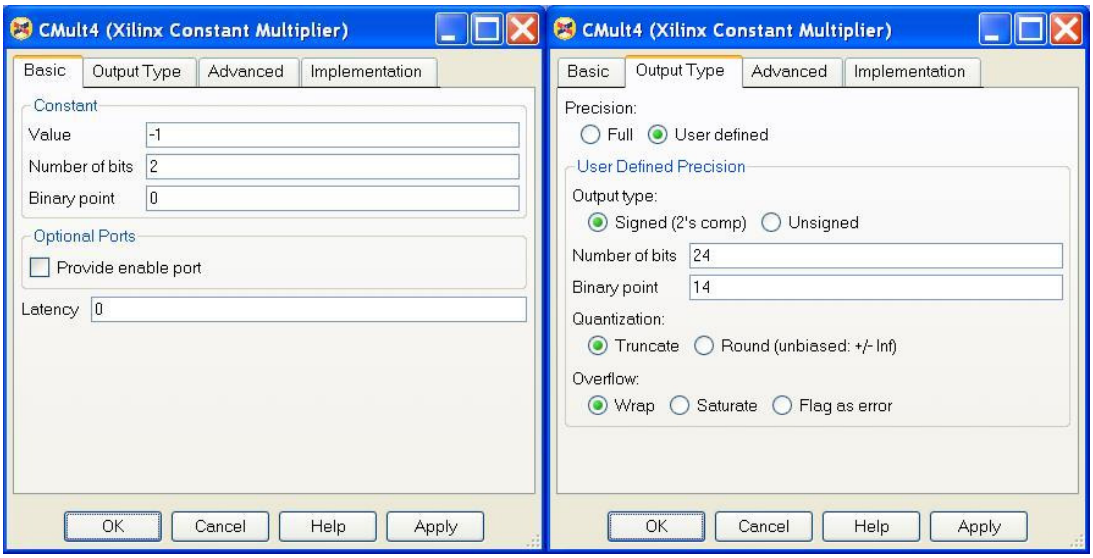


Figura 6.32: Cuadro de configuración del bloque "CMult"

Las salidas de los bloques comentados anteriormente, están conectadas las entradas a un subsistema que realiza un producto complejo (ver figura (6.28)). Este subsistema está compuesto por 4 multiplicadores, un restador y un sumador, conectados de tal manera que se obtiene el resultado del producto complejo (ver figura (6.33)). Estos bloques son bloques "***Black Box***", los cuales tienen asociado un código VHDL que podemos encontrar en el anexo B.

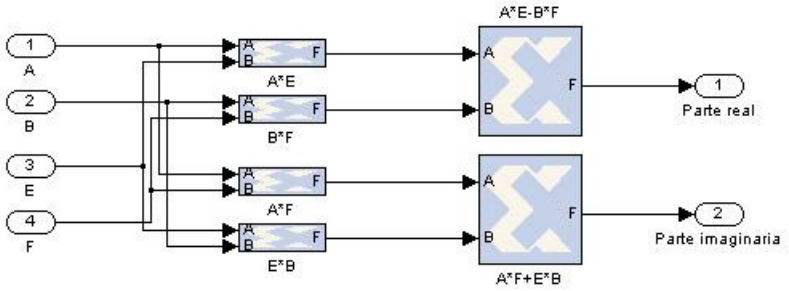


Figura 6.33: Interior del subsistema "Producto complejo2"

4) Pone en paralelo las dos mitades de la trama recibida y realiza el producto complejo del conjugado de la primera mitad de la trama recibida por la segunda mitad de la trama recibida (antena 2).

   Esta parte del diseño esta realizada de la misma manera que la descrita en el apartado anterior, pero referida a la antena receptora 2. Por tanto, no es necesaria su descripción.

5) Hace la suma de los datos de la trama de entrada (antena 1)

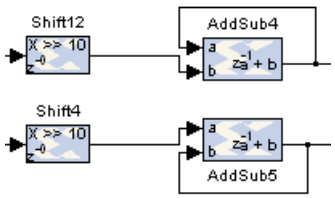


Figura 6.34: Etapa 5: "Hace la suma de los datos de la trama de entrada (antena 1)"

Esta etapa (figura (6.34)), realiza el sumatorio de la expresión (6.10), que recordamos a continuación particularizada para la antena 1:

$$\hat{\epsilon}_{frac_1} = angle\left(\sum_{t=0}^{127}\left(\underline{y}_t(1,1)\,\underline{y}_t(1,1)'^{*}\right)^{*}\left(\underline{y}_t(1,2)\,\underline{y}_t(1,2)'^{*}\right)\right) \qquad (6.21)$$

Sumatorio a realizar sobre 128 muestras

Para ello, se realiza una suma recursiva de los datos de entrada. Se utiliza un sumador que tiene su salida registrada, la cual es realimentada a la entrada del sumador. Así, el resultado de la suma, en cada ciclo de reloj, se registra a la salida y se realimenta a la entrada para sumarla

con los nuevos datos que llegan a una de las entradas del sumador. Como se puede observar en figura (6.34), esto se debe hacer tanto para la línea por donde se transmite la parte real de los datos, como para línea por donde se transmite la parte imaginaria, por lo que será necesario el uso de dos sumadores. Estos sumadores están configurados según la figura (6.35). Notar cómo se ha introducido *latencia* 1, para registrar un ciclo de reloj la salida del sumador.

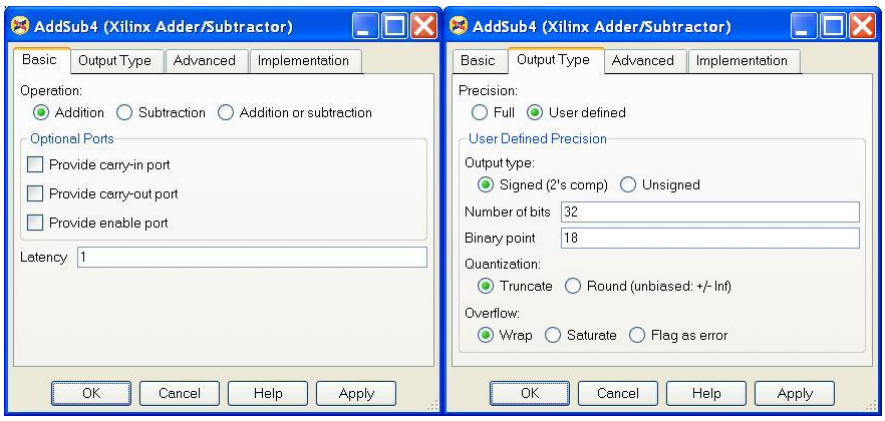


Figura 6.35: Cuadro de configuración de los bloques "AddSub" (Sumadores)

Cabe destacar que los datos de entrada a esta etapa, provenientes del producto complejo de la etapa anterior, poseen un valor elevado y al realizar el sumatorio de la expresión (6.21) puede que se sobrepase el número de bits establecidos para representar la información. Así, se ha decidido adaptar los datos de entrada mediante el empleo de bloques "***Shift***" (ver figura (6.34)). Estos bloques realizan un desplazamiento lógico a la derecha de 10 bits, lo que equivale a dividir en decimal entre 1024. Esto se hace así, para que el resultado del sumatorio no exceda el máximo de bits empleados, es decir, 24 bits. En la figura (6.36), se ilustra el cuadro de configuración de los bloques "***Shift***" empleados.

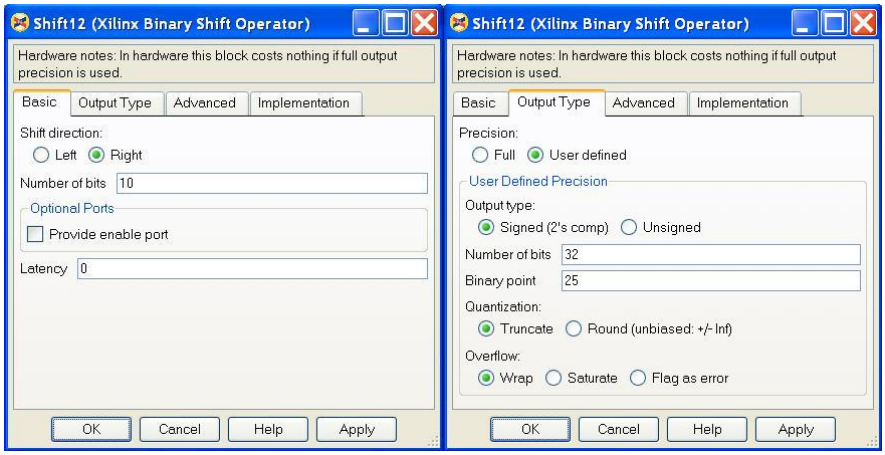


Figura 6.36: Cuadro de configuración de los bloques "Shfit"

6) Hace la suma de los datos de la trama de entrada (antena 2)

   Esta parte del diseño esta realizada de la misma manera que la descrita en el apartado anterior, pero referida a la antena receptora 2. Por tanto, no es necesaria su descripción.

7) Realiza la suma de los datos obtenidos para la antena 1 y para la antena 2

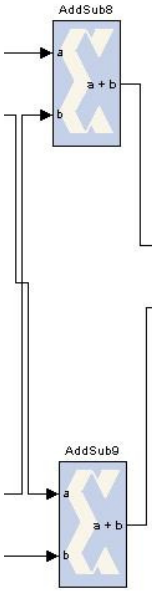


*Figura 6.37: Etapa 7: "*Realiza la suma de los datos obtenidos para la antena 1 y para la antena 2*"*

Esta etapa del diseño (figura (6.37)), trata de reproducir la suma de la expresión (6.11) que podemos encontrar en el apartado donde se explicó teóricamente el algoritmo de sincronización en frecuencia. Esta expresión es:

$$\hat{\epsilon}_{frac} = angle\left(\sum_{t=0}^{127}\left(\underline{y}_t(1,1)\,\underline{y}_t(q,1)'^{*}\right)^{*}\left(\underline{y}_t(1,2)\,\underline{y}_t(q,2)'^{*}\right)+\sum_{t=0}^{127}\left(\underline{y}_t(2,1)\,\underline{y}_t(2,1)'^{*}\right)^{*}\left(\underline{y}_t(2,2)\,\underline{y}_t(2,2)'^{*}\right)\right) \quad (6.22)$$

Suma realizada en esta etapa

La suma de la expresión (6.22), implica sumar los datos de la rama del diseño relativa a la antena 1 y los datos de la rama del diseño relativa a la antena 2. Así, simplemente esta etapa consta de dos sumadores, uno para sumar las partes reales y otro para sumar las partes imaginarias de los datos obtenidos en cada rama (ver figura (6.37)). Estos dos sumadores, están configurados según la figura (6.38).

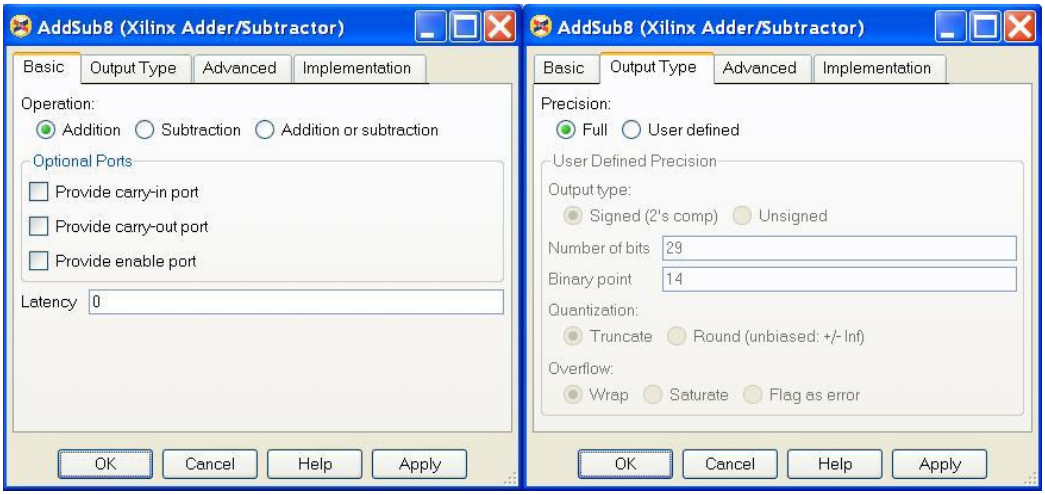


Figura 6.38: Cuadro de configuración de los bloques "AddSub" (Sumadores)

8) Realiza el cálculo de la fase

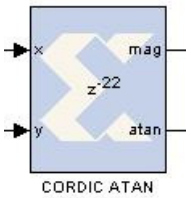


Figura 6.39: Etapa 8: "Realiza el cálculo de la fase"

Esta etapa del diseño (figura (6.39)), trata de reproducir la operación "*angle*" de la expresión (6.11) que podemos encontrar en el apartado donde se explicó teóricamente el algoritmo de sincronización en frecuencia. Esta expresión es:

$$\hat{\epsilon}_{frac} = angle\left(\sum_{t=0}^{127}\left(\underline{y}_t(1,1)\,\underline{y}_t(q,1)'^{*}\right)^{*}\left(\underline{y}_t(1,2)\,\underline{y}_t(q,2)'^{*}\right) + \sum_{t=0}^{127}\left(\underline{y}_t(2,1)\,\underline{y}_t(2,1)'^{*}\right)^{*}\left(\underline{y}_t(2,2)\,\underline{y}_t(2,2)'^{*}\right)\right) \qquad (6.23)$$

Operación a realizar en esta etapa

Esta operación "*angle*" implica calcular la fase de los datos calculados en etapas anteriores. Para ello, se emplea un bloque "***CORDIC ATAN***" donde la parte real de los datos se introduce por la entrada "*x*" del bloque, y los datos de la parte imaginaria por la entrada "*y*", tal y como es muestra en la figura (6.39). Este bloque posee dos salidas, sin embargo, solo nos interesará la salida "*atan*", la cual nos dará el resultado de la fase a calcular según la expresión (6.23). Este bloque está configurado según la figura (6.40).

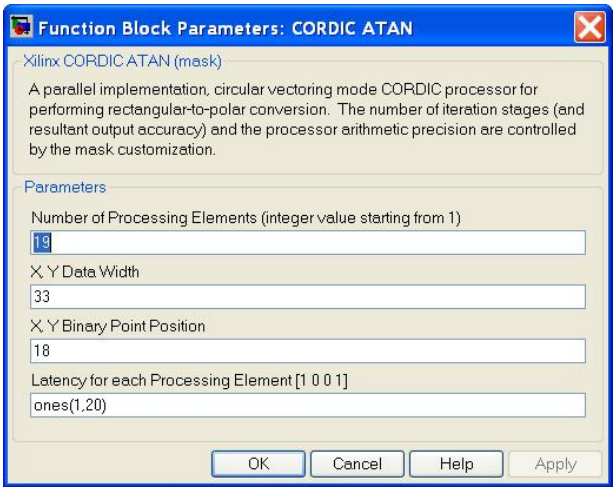


Figura 6.40: Cuadro de configuración del bloque "CORDIC ATAN"

4.1.4 Bloque 4: "Corrector del "*offset*" frecuencial"

En la figura (6.41), mostramos el aspecto general que tiene este bloque del diseño. Sobre ella, resaltaremos las partes más importantes mediante recuadros y las numeraremos para describirlas más adelante.

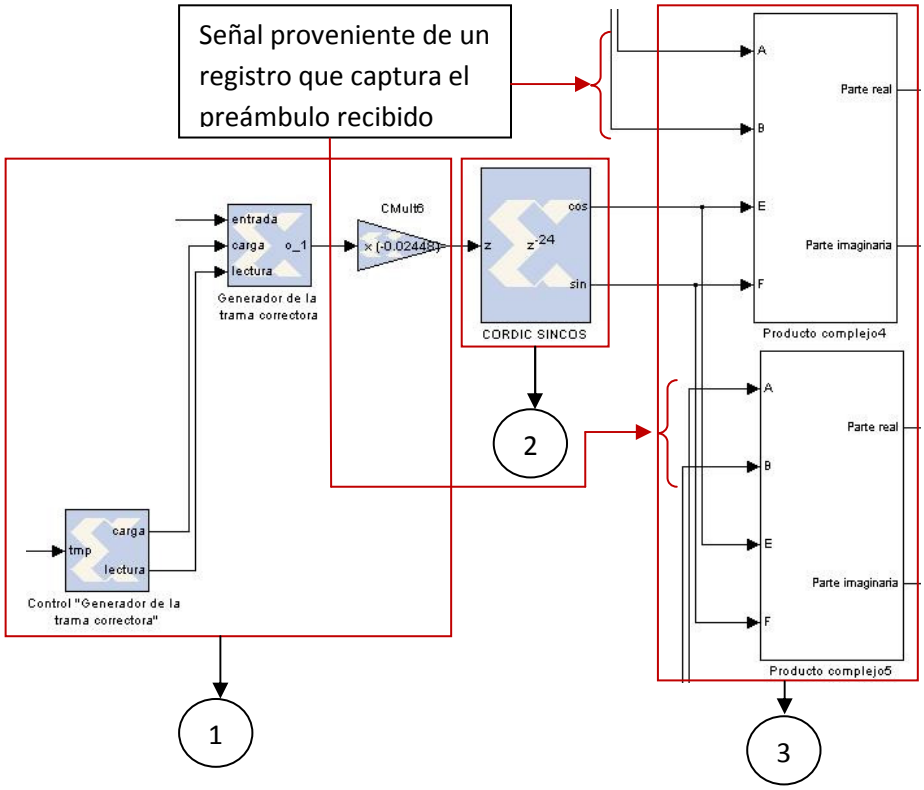


Figura 6.41: Bloque 4: "Corrector del "offset" frecuencial"

Corregir el efecto del desplazamiento en frecuencia se puede modelar según la siguiente expresión:

$$p_{correct}(n) = p_{recib}(n) \cdot e^{\frac{j2\pi(-\hat{\varepsilon}_{fraq})n}{N}} \qquad (6.24)$$

donde $p_{recib}(n)$ es el preámbulo recibido en el dominio del tiempo y $e^{\frac{j2\pi(-\hat{\varepsilon}_{fraq})n}{N}}$ es una exponencial compleja cuya fase depende del desplazamiento de frecuencia estimado en la etapa anterior. Esta operación deberá realizarse para el preámbulo recibido en la antena 1 y para el preámbulo recibido en la antena 2. Así, el objetivo del bloque que detallaremos a continuación es modelar en hardware la expresión (6.24), la cual corrige el "*offset*" frecuencial.

1) Generación de la fase $\frac{2\pi(-\hat{\varepsilon}_{fraq})n}{N}$

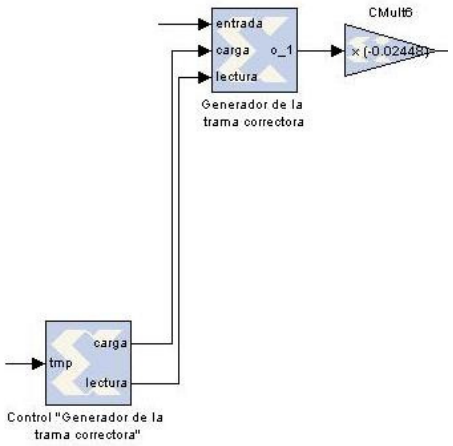


Figura 6.42: Etapa 1: "Generación de la fase $\frac{2\pi(-\hat{\varepsilon}_{fraq})n}{N}$*"*

Esta etapa del diseño (figura (6.42)), calcula la fase de la exponencial compleja de la expresión (6.24). Esta fase se debe aplicar a cada muestra del preámbulo recibido, por tanto, se deberá crear una secuencia de 256 muestras que represente la fase de la exponencial compleja de la expresión (6.24). Así, se ha ideado mediante un bloque "***Black Box***", identificado bajo el nombre "*Generador de la trama correctora*" (ver figura (6.42)), un circuito que realiza la siguiente función:

- Cuando la entrada "$carga$" esta activa, el bloque registra el dato que tiene en la entrada "$entrada$". Este dato será el desplazamiento en frecuencia calculado en la etapa anterior.
- Cuando la entrada "$lectura$" está activa, el bloque lee del registro donde está el valor de la estimación del desplazamiento en frecuencia y lo multiplica por una secuencia generada por un contador interno de 0 a 255. El resultado de esta multiplicación se saca por la salida "o_1", dando lugar a la secuencia deseada ($\hat{\varepsilon}_{fraq}n$).

El bloque "*Generador de la trama correctora*", esta modelado mediante un bloque "***Black Box***", el cual tiene un código VHDL asociado que podemos encontrar en el anexo B. Su cuadro de configuración se ilustra en la figura (6.43).

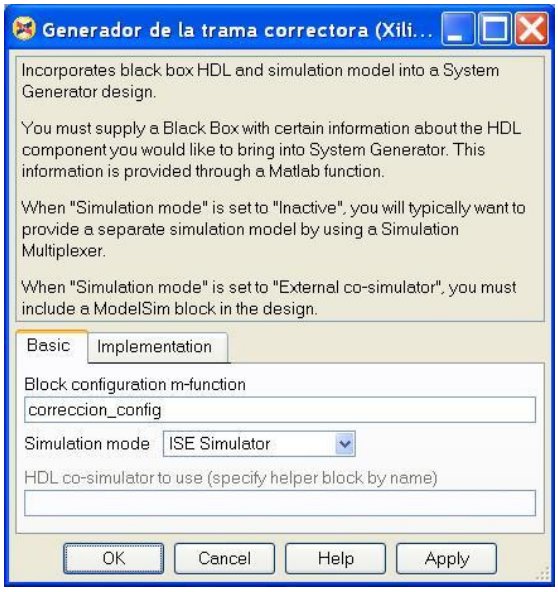


Figura 6.43: Cuadro de configuración del bloque "Black Box" utilizado para modelar el bloque que genera la trama correctora

La secuencia de salida del bloque "*Generador de la trama correctora*" debe de ser multiplicada por el factor $-\frac{2\pi}{N}$, que en nuestro caso es $-\frac{2\pi}{N} = -\frac{2\pi}{256} \cong -0.0245$. Esto se ha realizado mediante un bloque "***CMult***", conectado según la figura (6.42), el cual está configurado según la figura (6.44).

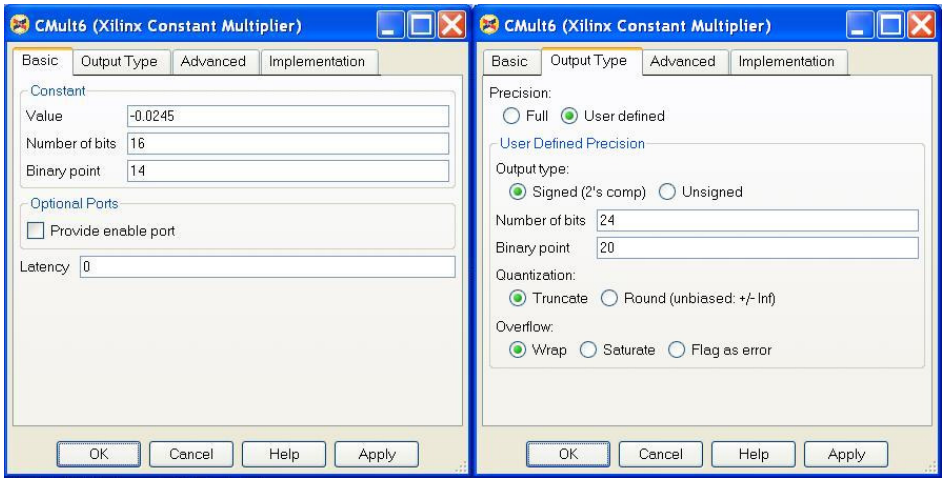


Figura 6.44: Cuadro de configuración del bloque "CMult"

2) Generación de la exponencial compleja $e^{\frac{j2\pi(-\hat{\varepsilon}_{fraq})n}{N}}$

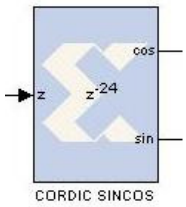


Figura 6.45: Etapa 2: "Generación de la exponencial compleja $e^{\frac{j2\pi(-\hat{\varepsilon}_{fraq})n}{N}}$

Esta etapa del diseño (figura (6.45)), genera la secuencia de exponenciales complejas de la expresión (6.24) a partir de la secuencia de datos generados en la etapa anterior. Para ello, se ha utilizado un bloque que se basa en la regla de "*Euler*" para números complejos. Según "*Euler*", una exponencial compleja de este tipo se puede expresar de la forma:

$$e^{j\theta} = \cos\theta + \mathrm{j}\cdot\sin\theta \qquad (6.25)$$

que aplicada al caso que nos ocupa, se tiene:

$$e^{\frac{j2\pi(-\hat{\varepsilon}_{fraq})n}{N}} = \cos\left(\frac{2\pi(-\hat{\varepsilon}_{fraq})n}{N}\right) + \mathrm{j}\cdot\sin\left(\frac{2\pi(-\hat{\varepsilon}_{fraq})n}{N}\right) \qquad (6.26)$$

Así, la operación definida en la expresión (6.26), la realizamos mediante un bloque "***CORDIC SINCOS***", que nos devuelve en dos líneas, el coseno y el seno de la secuencia que tiene a la entrada, proveniente de la etapa anterior (fase de la secuencia de exponenciales complejas). Este bloque está configurado según la figura (6.46). De esta manera, hemos generado la parte real e imaginara de la secuencia que nos servirá para corregir el "*offset*" en frecuencia del preámbulo recibido. Así, solo habrá que realizar, en la última etapa, el producto complejo de esta secuencia con el preámbulo recibido en ambas antenas receptoras para corregir el "*offset*" frecuencial.

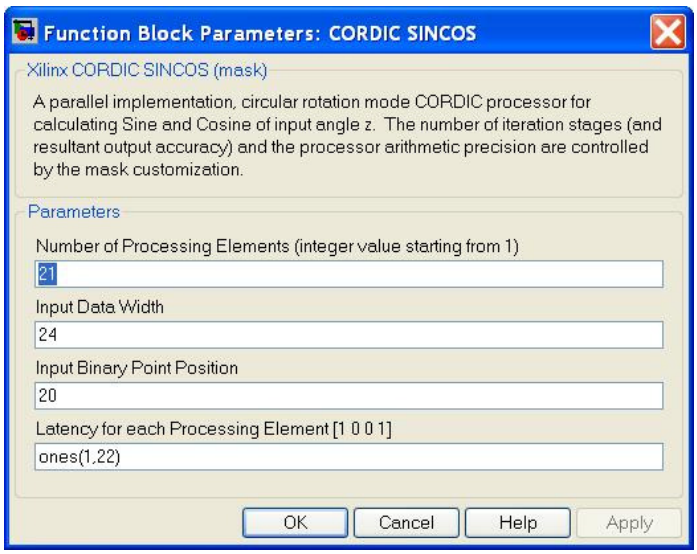


Figura 6.46: Cuadro de configuración del bloque "CORDIC SINCOS"

3) Producto complejo entre el preámbulo recibido y la exponencial compleja calculada

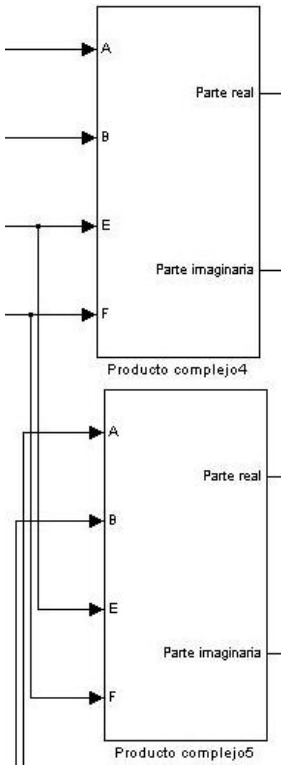


Figura 6.47: Etapa 3: "Producto complejo entre el preámbulo recibido y la exponencial compleja calculada"

En esta etapa del diseño (figura (6.47)), únicamente realiza el producto complejo del preámbulo recibido en cada antena receptora por la secuencia generada en la etapa anterior. Así, esta etapa solo está compuesta por dos subsistemas ("*Producto complejo4" y "Producto complejo5"*), que realizan tal producto complejo. En cada subsistema, se realiza el producto complejo de la misma manera que comentamos en otros apartados, por lo que solo mostraremos en la figura (6.48) como están hechas las conexiones de los distintos bloques que intervienen en el cálculo del producto complejo.

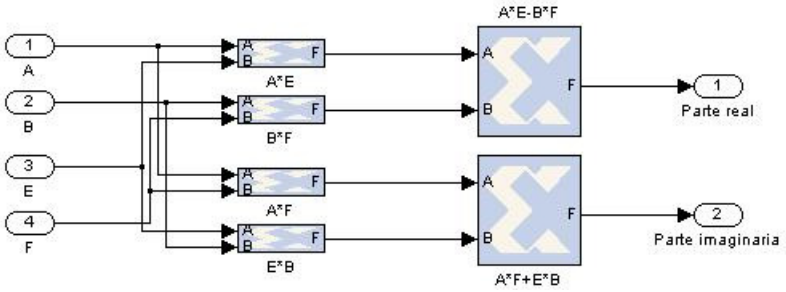


Figura 6.48: Interior del subsistema "Producto complejo4"

Dos de las entradas de los subsistemas "*Producto complejo*", están conectadas a los registros donde se almacena el preámbulo recibido (ver figura (6.41)). El cometido de estos registros ya se explicó en el apartado (4.1.3). Únicamente, comentar que el preámbulo es leído de la segunda salida de estos bloques de registro.

Por último, el preámbulo corregido se vuelve a pasar por el módulo de estimación de canal para reestimar el canal, una vez se ha corregido el desplazamiento en frecuencia. Esto supone el empleo de una etapa al comienzo de la estimación de canal constituida por multiplexores de dos entradas (ver visión general del módulo de estimación de canal, figura (5.3)). Estos multiplexores están configurados según la figura (6.49). La entrada "*sel*" de los bloques "***Mux***" es la encargada de seleccionar una de las dos entradas que tiene un multiplexor. La señal en "*sel*", es una señal de control generada por un bloque "***Black Box***", identificado bajo el nombre "*señal control mux*", el cual está, a su vez, controlado por el contador común ya descrito en apartados anteriores. Cuando el contador común llega a una determinada cuenta, el bloque "*señal control mux*" genera una señal que hace cambiar la posición de los multiplexores,

propiciando la entrada del preámbulo recibido con el "*offset*" frecuencial corregido, al módulo de estimación de canal.

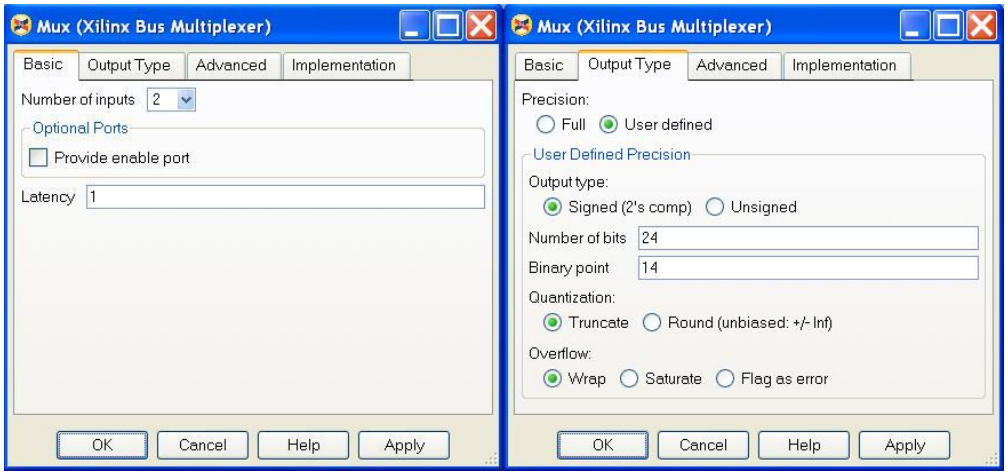


Figura 6.49: Cuadro de configuración de los bloques "Mux"

El bloque "*señal control mux*", tiene un código VHDL asociado el cual podemos encontrar en el anexo B. El cuadro de configuración de este bloque se ilustra en la figura (6.50).

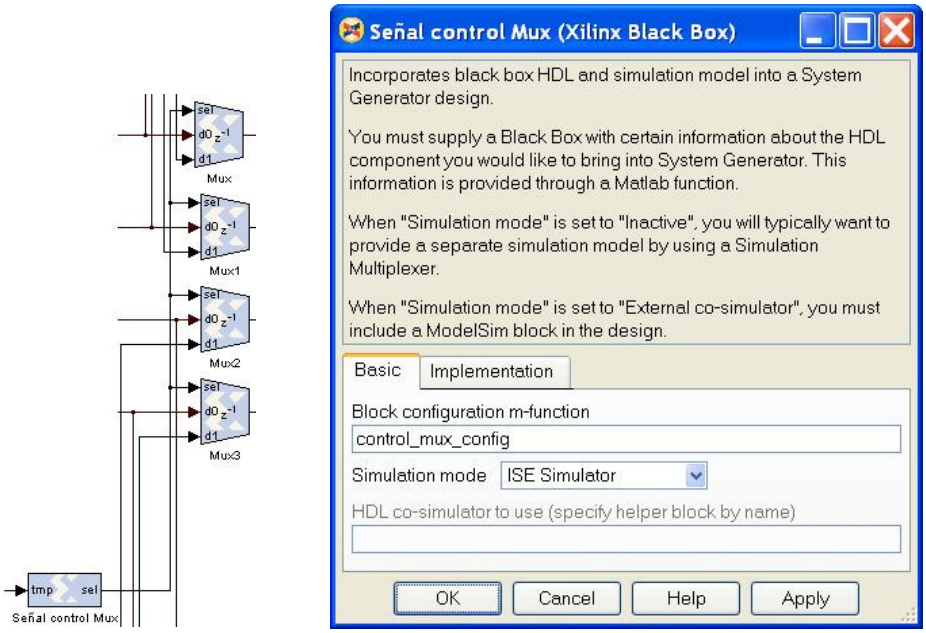


Figura 6.50: Apariencia de la etapa de Multiplexores y cuadro de configuración del bloque "Box Black" empleado para la modelación del bloque "señal control mux"

4.2 Módulo de Sincronización Temporal

El diseño del módulo que desempeña el algoritmo de sincronización temporal, ha sido realizado utilizando bloques de la librería *Xilinx Blockset de Xilinx System Generator for DSP®* [4].Estos bloques fueros descritos en el capítulo 4. El algoritmo que se ha descrito en hardware se corresponde con el algoritmo propuesto en [3], que anteriormente explicamos en el apartado (3.). El artículo [3], propone una arquitectura para el diseño hardware del módulo de sincronización temporal. Así, realizaremos el diseño propuesto en [3] en Simulink®. El diseño que se propone, ilustra en la figura (6.49) donde se destacan las principales etapas.

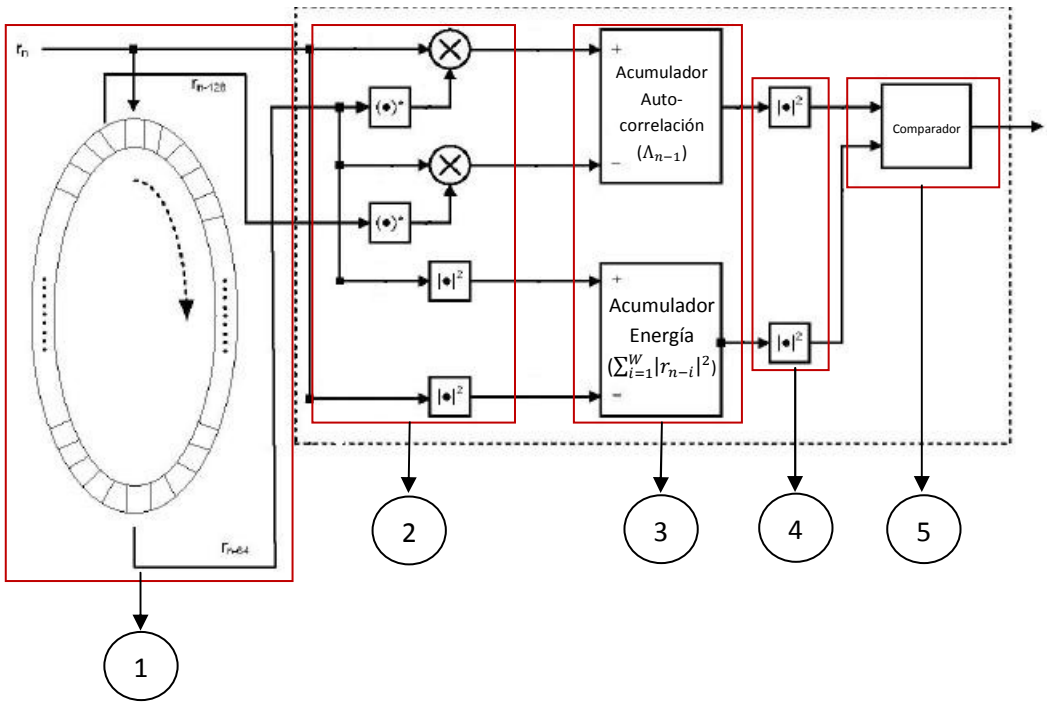


Figura 6.51: Módulo de sincronización temporal [3]

Antes de comenzar con la descripción del módulo de sincronización temporal, es preciso tener en cuenta, previamente, una serie de consideraciones generales:

- Para comprobar el correcto funcionamiento del hardware diseñado, es preciso generar las señales de entrada a dicho hardware. Las señales de entrada deberán de corresponderse con el preámbulo recibido, el cual habrá sufrido un desplazamiento temporal. Para ello, se ha

simulado en Matlab® el paso del preámbulo a través del canal, con dicho desplazamiento temporal. La función de Matlab® que simula el paso del preámbulo por el canal es "*Timing.m*", la cual realiza una llamada a otra función de Matlab® ("*chan_SUI.m*") que simula uno de los 4 canales SISO del canal 2x2 MIMO. Esta función, nos devolverán la parte real y la parte imaginaria de los preámbulos con desplazamiento temporal en cada antena receptora en los vectores "*En_re_desplazado1" y "En_im_desplazado1*". Seremos capaces de importar los datos guardados en esos vectores generados por la función de Matlab® mediante el uso de bloques "***From Workspace***".

- Las dos funciones de Matlab®, anteriormente comentadas, las podemos encontrar en el Anexo A.
- El modelo de canal simulado en Matlab® es el mismo que empleamos en el capítulo de estimación de canal.
- Para definir los puertos de entrada al hardware diseñado, se han utilizado bloques "***Gateway In***" configurados para trabajar con datos tipo "*signed*" de 24 bits con el punto binario en el bit 14. Esto se ha realizado así, para poder representar de una manera suficientemente precisa la información. Este número de bits se trata de mantener a lo largo del diseño.

➔ *Visión general del modelo Simulink®*

En la figura (6.52), mostramos el aspecto general que tiene este módulo de sincronización temporal. Sobre ella, resaltaremos las partes más importantes mediante recuadros y las numeraremos para describirlas más adelante. Aquí, se puede observar la relación entre el modelo Simulink® realizado y la arquitectura propuesta en [3] ilustrada en la figura (6.49).

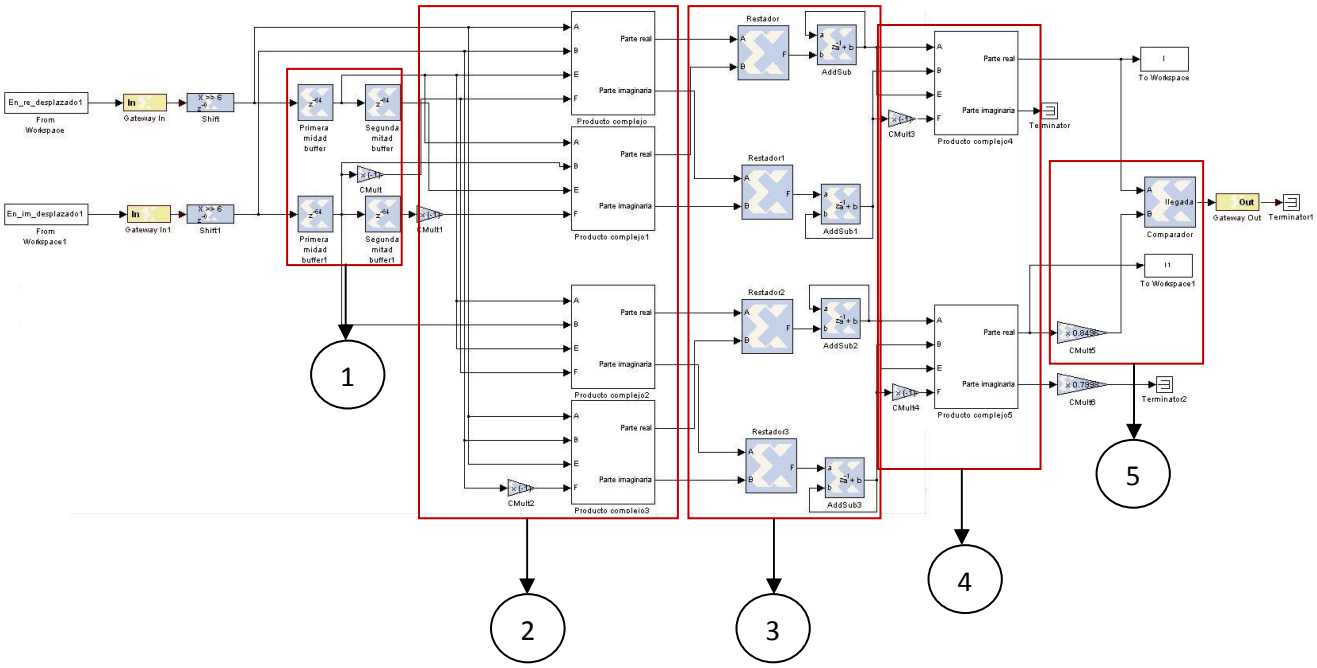


Figura 6.52: Visión general del Módulo de sincronización temporal

4.2.1 Etapa 1: "*Buffer*" con capacidad para 128 muestras

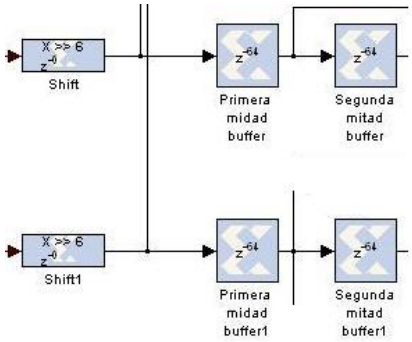


Figura 6.53: Etapa 1: "Buffer" con capacidad para 128 muestras

Esta etapa del diseño (ver figura (6.53)), trata de realizar el "*buffer*" circular propuesto en [3], el cual podemos ver en la figura (6.49). El preámbulo recibido contiene una parte real y una parte imaginaria, por lo que este "*buffer*" se ha

ideado en dos líneas, una para la parte real y otra para la parte imaginaria (ver figura (6.51)). Como vemos en la figura (6.51), los datos se leen del "*buffer*", en la mitad del "*buffer*" y en la parte final del "*buffer*". Así, se ha decidido modelar el "*buffer*" mediante una serie de bloques "***Delay***" que retardan los datos de entrada 64 ciclos de reloj. El primer bloque "***Delay***" de cada línea (real e imaginaria) modela la primera mitad del "*buffer*" y el segundo bloque "***Delay***" puesto en serie, modela la segunda mitad del "*buffer*". Los bloques "***Delay***" utilizados están configurados según la figura (6.54).

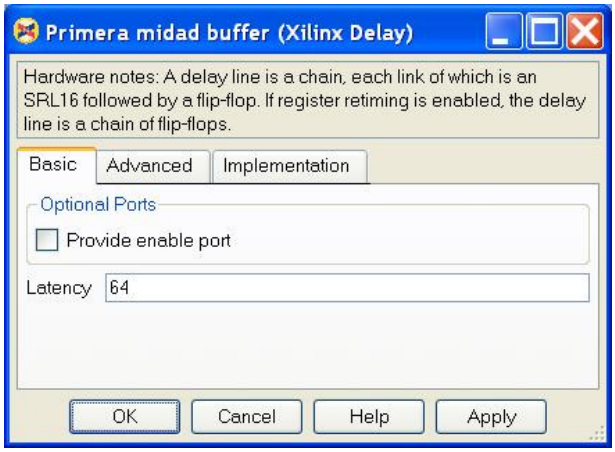


Figura 6.54: Cuadro de configuración de los bloques "Delay"

Como vemos en la figura (6.53), a la entrada de esta etapa aparecen dos bloques "***Shift***". Esto es debido a que el algoritmo de sincronización temporal tendrá que realizar una suma recursiva, la cual puede dar un valor demasiado elevado que no se pueda representar mediante 24 bits. Lo que hacen estos bloques "***Shift***" es hacer un desplazamiento lógico de 6 bits a la derecha de los datos de entrada, o lo que es lo mismo, dividirlos entre 64, de manera que la suma recursiva que requiere al algoritmo (se hará más adelante) no sobre pasa un valor que no se pueda representar con 24 bits. Estos bloques "***Shift***" están configurados según la figura (6.55).

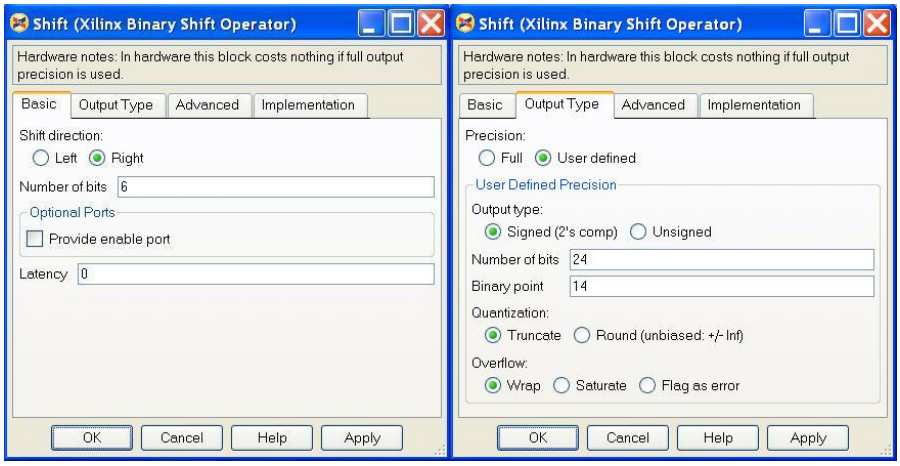


Figura 6.55: Cuadro de configuración de los bloques "Shift"

4.2.2 Etapa 2: Productos complejos

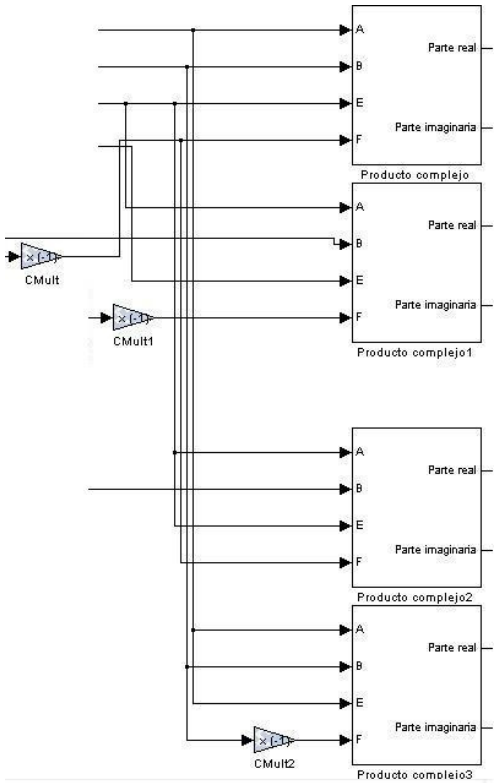


Figura 6.56: Etapa 2: Productos complejos

En esta etapa del diseño (ver figura (6.56)), se realizan una serie de productos complejos de los datos leídos del "*buffer*" de la etapa anterior. Estos productos complejos son realizados por una serie de subsistemas, tal y como podemos ver en la figura (6.56). Estos subsistemas realizan un producto complejo de la misma manera que comentamos en otros apartados, por lo que simplemente mostraremos en la figura (6.57) cuál es el contenido de estos subsistemas (4 multiplicadores, un restador y un sumador debidamente conectados para hacer un producto complejo).

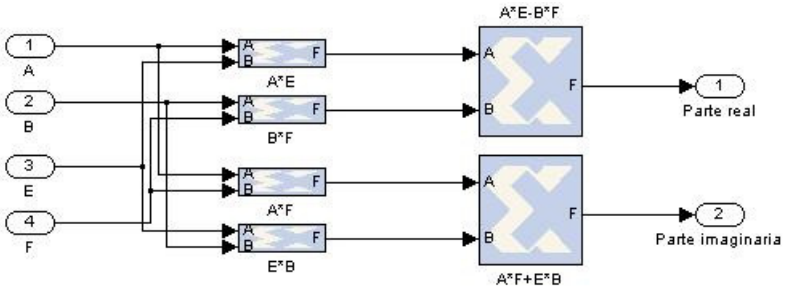


Figura 6.57: Subsistema "Producto complejo"

Los dos primeros subsistemas *("Producto complejo" y "Producto complejo1")*, se encargan de realizar el producto complejo de la fórmula iterativa (6.13), la cual recordamos a continuación:

$$\Lambda_n = \Lambda_{n-1} + r_{n-1} \cdot r^*_{n-W} - r_{n-W} \cdot r^*_{n-2W} \qquad (6.27)$$

Productos complejos

Estos productos complejos se muestran de forma gráfica mediante subsistemas en la figura (6.56), la cual hemos seguido para hacer el diseño.

Por otra parte, los dos últimos subsistemas *("Producto complejo2" y "Producto complejo3")* se encargan de realizar las operaciones del módulo al cuadrado de la fórmula iterativa (6.14), cual recordamos a continuación:

$$E_n = \sum_{i=1}^{W} |r_{n-i}|^2 = E_{n-1} + |r_{n-W}|^2 - |r_{n-1}|^2 \qquad (6.28)$$

Hacer el módulo al cuadrado de un número complejo, implica multiplicar ese mismo número complejo por su conjugado. Por lo que la expresión (6.28), la podemos expresar de la siguiente manera:

$$E_n = \sum_{i=1}^{W} |r_{n-i}|^2 = E_{n-1} + r_{n-W} \cdot r_{n-W}^* - r_{n-1} \cdot r_{n-1}^* \qquad (6.29)$$

Productos complejos

Así, los subsistemas *"Producto complejo2" y "Producto complejo3"*, están conectados al "*buffer*" de la etapa anterior de manera que realizan los productos complejos de la expresión (6.29).

Como vemos de las expresiones (6.27) y (6.29), hay factores de los productos complejos que están conjugados. Esto implica negar las partes imaginarias de estos factores. Esto se realiza en el modelo Simulink® mediante bloques "***CMult***" los cuales multiplican las partes imaginarias por -1, tal y como se observa en la figura (6.56). Estos bloques están configurados según la figura (6.58).

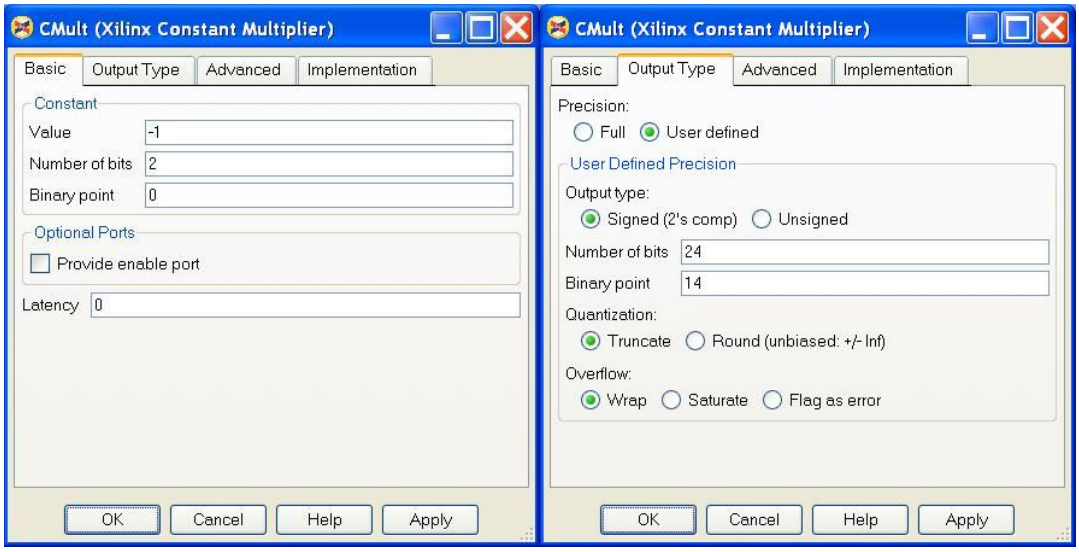


Figura 6.58: Cuadro de configuración de los bloques "CMult"

4.2.3 Etapa 3: Resta y acumulación

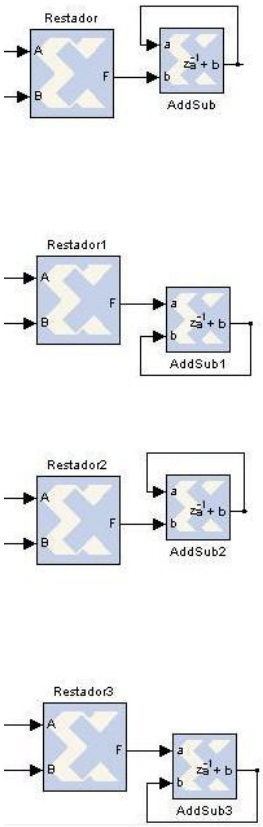


Figura 6.59: Etapa 3: Resta y acumulación

Esta etapa del diseño (figura (6.59)), se encarga de realizar las operaciones de resta y acumulación de las fórmulas iterativas (6.27) y (6.29), las cuales recordamos a continuación:

$$\Lambda_n = \Lambda_{n-1} + r_{n-1} \cdot r_{n-W}^* - r_{n-W} \cdot r_{n-2W+1}^* \qquad (6.30)$$

Suma recursiva (acumulación)

Resta de productos complejos

$$E_n = \sum_{i=1}^{W} |r_{n-i}|^2 = E_{n-1} + r_{n-W} \cdot r_{n-W}^* - r_{n-1} \cdot r_{n-1}^* \qquad (6.31)$$

Para realizar la resta se utilizan bloques restadores, tal y como es muestra en la figura (6.59). Estos bloques restador, están modelados mediante bloques "***Black Box***", los cuales tienen asociados un código VHDL que podemos

encontrar en el anexo B. El cuadro de configuración de estos bloques se muestra en la figura (6.60).

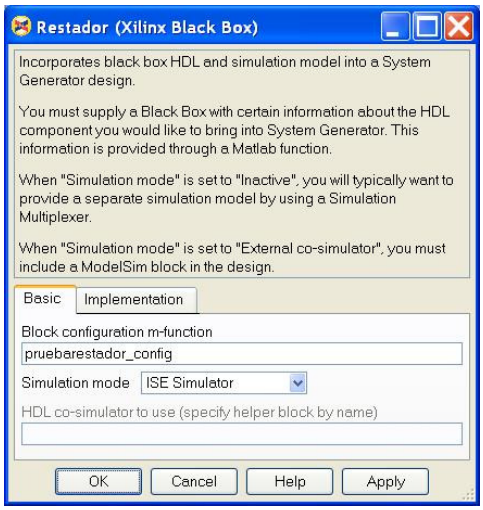


Figura 6.60: Cuadro de configuración de los bloques "Black Box" empleados para modelar los restadores

Por otra parte, la suma recursiva (*acumulación*) se realiza mediantes bloques "***AddSub***" configurados como sumadores. Cada sumador, tiene su salida registrada, la cual es realimentada a la entrada del sumador. Así, el resultado de la suma, en cada ciclo de reloj, se registra a la salida y se realimenta a la entrada para sumarla con los nuevos datos que llegan a una de las entradas del sumador. Estos sumadores están configurados según la figura (6.61). Notar cómo se ha introducido latencia 1, para registrar un ciclo de reloj la salida del sumador.

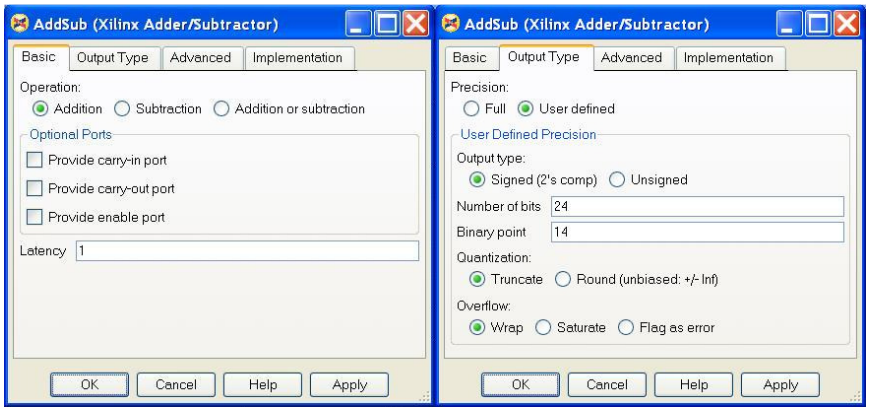


Figura 6.61: Cuadro de configuración de los bloques "AddSub"

4.2.4 Etapa 4: Módulo al cuadrado

Figura 6.62: Etapa 4: Módulo al cuadrado

Esta etapa del diseño (figura (6.62)), calcula el numerador y el denominador de la métrica (6.12), la cual recordamos a continuación:

$$\mu_n = \frac{\left|\sum_{i=1}^{W} r_{n-i} \cdot r_{n-i-W}^{*}\right|^2}{\left|\sum_{i=1}^{W} |r_{n-i}|^2\right|^2} \qquad (6.32)$$

Datos calculados en la etapa anterior

Para ello, únicamente será necesario de realizar el módulo al cuadrado de los datos calculados en la etapa anterior. Para realizar el módulo al cuadrado, utilizamos dos subsistemas que modelan un producto complejo, los cuales debidamente conectados, realizan el módulo al cuadrado de los datos que les llegan a sus entradas (el módulo al cuadrado de un número complejo se calcula haciendo el producto complejo de ese mismo número por su conjugado). Estos subsistemas son idénticos a los empleados en la etapa anterior, por lo que no será necesaria su descripción.

4.2.5 Etapa 5: Comparador

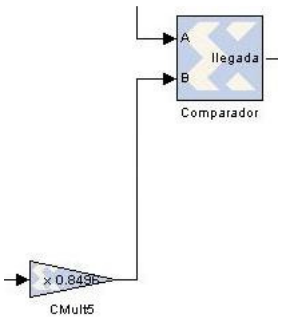


Figura 6.63: Etapa 5: Comparador

Esta etapa del diseño (figura (6.63)), realiza la operación de comparación definida en la expresión (6.15), la cual recordamos a continuación:

$$\frac{\left|\sum_{i=1}^{W} r_{n-i}\cdot r_{n-i-W}^{*}\right|^{2}}{\left|\sum_{i=1}^{W}|r_{n-i}|^{2}\right|^{2}} > th \rightarrow \left|\sum_{i=1}^{W} r_{n-i}\cdot r_{n-i-W}^{*}\right|^{2} - th\left|\sum_{i=1}^{W}|r_{n-i}|^{2}\right|^{2} > 0 \qquad (6.33)$$

Para ello, mediante el empleo de un bloque "***Black Box***", se ha modelado un bloque comparador que realiza la comparación definida en (6.33). Este bloque "***Black Box***", identificado bajo el nombre "*comparador*", tiene asociado un código VHDL asociado que podemos encontrar en el anexo B. El cuadro de configuración de este bloque se ilustra en la figura (6.64).

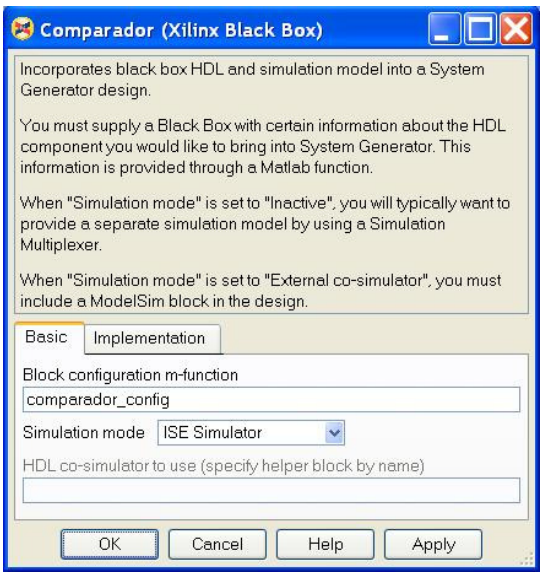


Figura 6.64: Cuadro de configuración del bloque "Black Box" empleado en el modelado del bloque comparador

El bloque "*comparador*", compara los datos generados en la etapa anterior de manera que cuando se cumple la condición (6.33), se genera a su salida una señal en estado alto que cuyo flanco de subida determina cuando se sobrepasa el umbral th, el cual determina el instante de llegada del preámbulo.

Notar como el denominador de la métrica (6.15), se encuentra multiplicado por el umbral th. Esto se realiza en el modelo Simulink ideado (ver figura (6.63)) mediante un bloque "***CMult***" configurado según la figura (6.65).

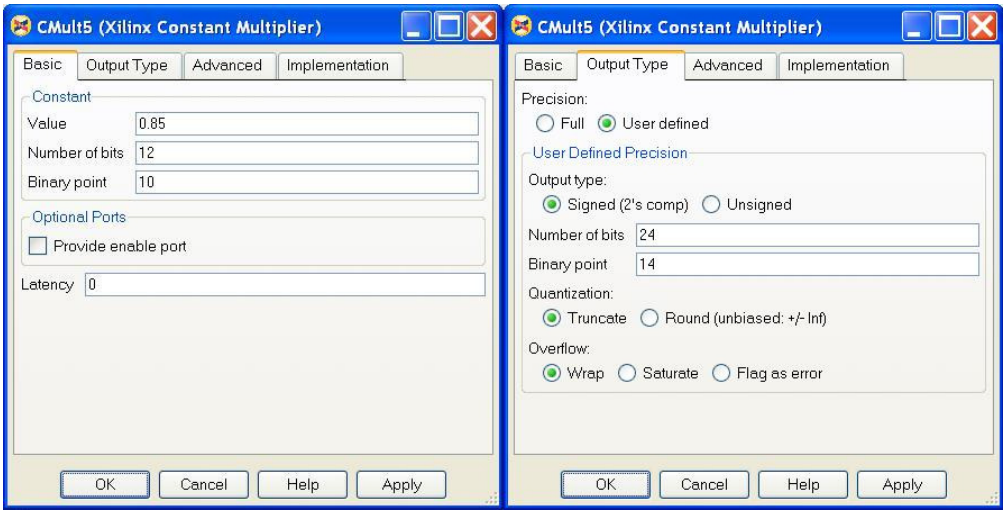


Figura 6.65: Cuadro de configuración del bloque "CMult"

5. Validación de los módulos diseñados

5.1 Validación del módulo de Sincronización en frecuencia

Para comprobar que el hardware diseñado para el algoritmo de sincronización frecuencial funciona correctamente, validaremos tal hardware en los puntos mostrados en la figura (6.66). Los resultados obtenidos en esos puntos serán mostrados mediante el empleo del bloque "***Wavescope***". Los resultados obtenidos de la simulación del modelo Simulink®, los cuales se mostrarán en la ventana de visualización de "***Wavescope***", serán comparados con los resultados que ofrece una función de Matlab® que simula el algoritmo de sincronización frecuencial. La función de Matlab® que simula el algoritmo de estimación de sincronización frecuencial es "*MLTF_offset.m*", la cual podemos encontrar en el anexo A.

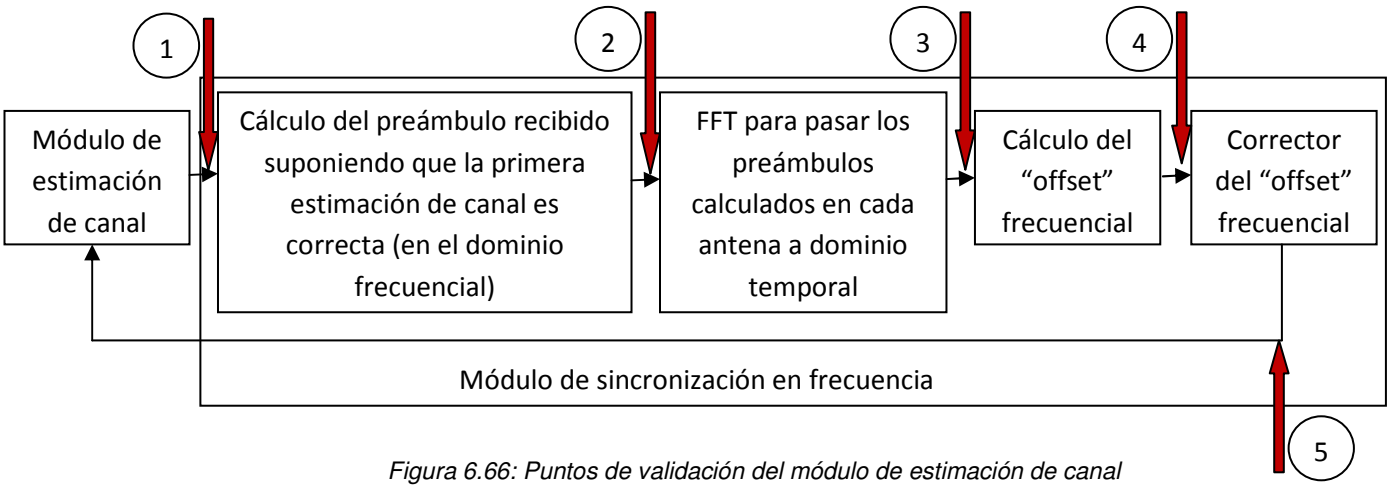


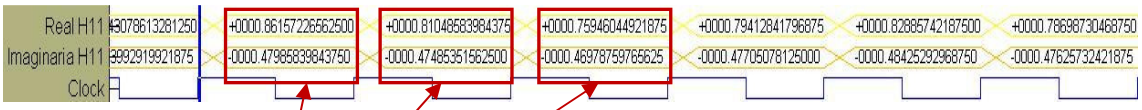
Figura 6.66: Puntos de validación del módulo de estimación de canal

Los puntos a evaluar según la figura (6.66) son:

1) Salida del bloque "*Módulo de estimación de canal*".
2) Salida del bloque "*Cálculo del preámbulo recibido suponiendo que la primera estimación de canal es correcta (en el dominio frecuencial)".*
3) Salida del bloque "*FFT para pasar los preámbulos calculados en cada antena a dominio temporal*".
4) Salida del bloque "*Cálculo del "offset" frecuencial*".
5) Salida del bloque "*Corrector del "offset" frecuencial*".

5.1.1 Salida del bloque "Módulo de estimación de canal"

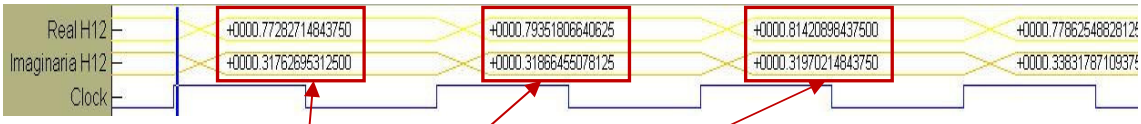
En este punto del diseño, vamos a comprobar que los canales estimados en el módulo de estimación de canal han sido calculados correctamente. Estos canales estimados serán las entradas al hardware diseñado para la sincronización en frecuencia. En las figuras (6.67), (6.68), (6.69) y (6.70), se ilustra las primeras muestras de los 4 canales estimados tanto en la simulación de Matlab® como la de Simulink® (solo se mostrarán las primeras subportadoras con información, omitiendo las bandas de guarda).




```
H_inter_11 =

         0
  0.8756 - 0.4876i
  0.8237 - 0.4825i
  0.7718 - 0.4774i
  0.8070 - 0.4847i
  0.8423 - 0.4921i
  0.7998 - 0.4839i
  0.7572 - 0.4758i
  0.7683 - 0.4827i
  0.7794 - 0.4897i
  0.7525 - 0.4587i
  0.7256 - 0.4277i
  0.7425 - 0.4309i
  0.7594 - 0.4342i
  0.7747 - 0.4748i
  0.7901 - 0.5154i
  0.8018 - 0.4859i
  0.8135 - 0.4564i
  0.7576 - 0.4611i
  0.7017 - 0.4658i
  0.7260 - 0.4623i
  0.7502 - 0.4587i
  0.6900 - 0.4768i
```

Figura 6.67: Comparativa entre la simulación de Matlab® y del modelo Simulink® para el Canal estimado H11




```
H_inter_12 =

         0
   0.7854 + 0.3228i
   0.8064 + 0.3239i
   0.8274 + 0.3249i
   0.7913 + 0.3438i
   0.7551 + 0.3628i
   0.7715 + 0.3139i
   0.7879 + 0.2651i
   0.7985 + 0.2855i
   0.8092 + 0.3059i
   0.7916 + 0.2644i
   0.7741 + 0.2228i
   0.7161 + 0.2310i
   0.6581 + 0.2392i
   0.7302 + 0.2634i
   0.8024 + 0.2875i
   0.7461 + 0.2932i
   0.6899 + 0.2990i
   0.7251 + 0.2591i
   0.7604 + 0.2192i
   0.7124 + 0.2233i
   0.6644 + 0.2273i
   0.6980 + 0.2129i
```

Figura 6.68: Comparativa entre la simulación de Matlab® y del modelo Simulink® para el Canal estimado H12

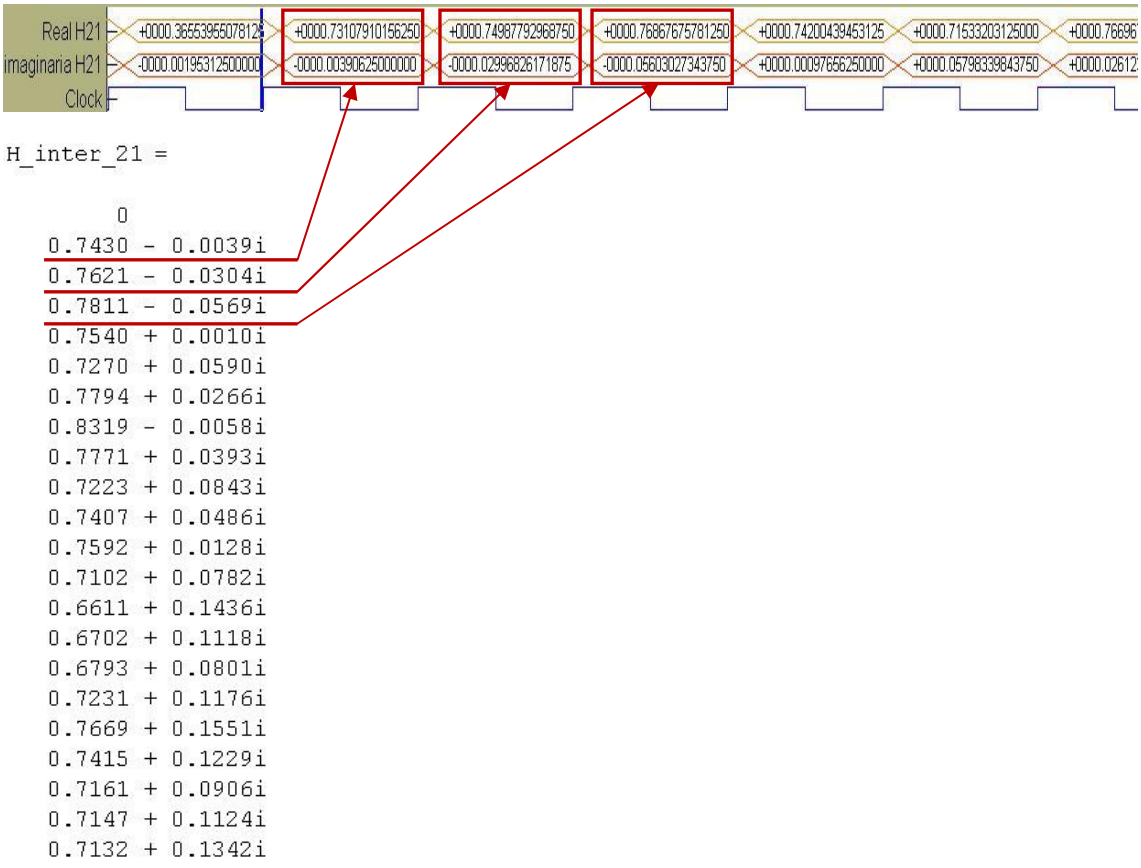


Figura 6.69: Comparativa entre la simulación de Matlab® y del modelo Simulink® para el Canal estimado H21

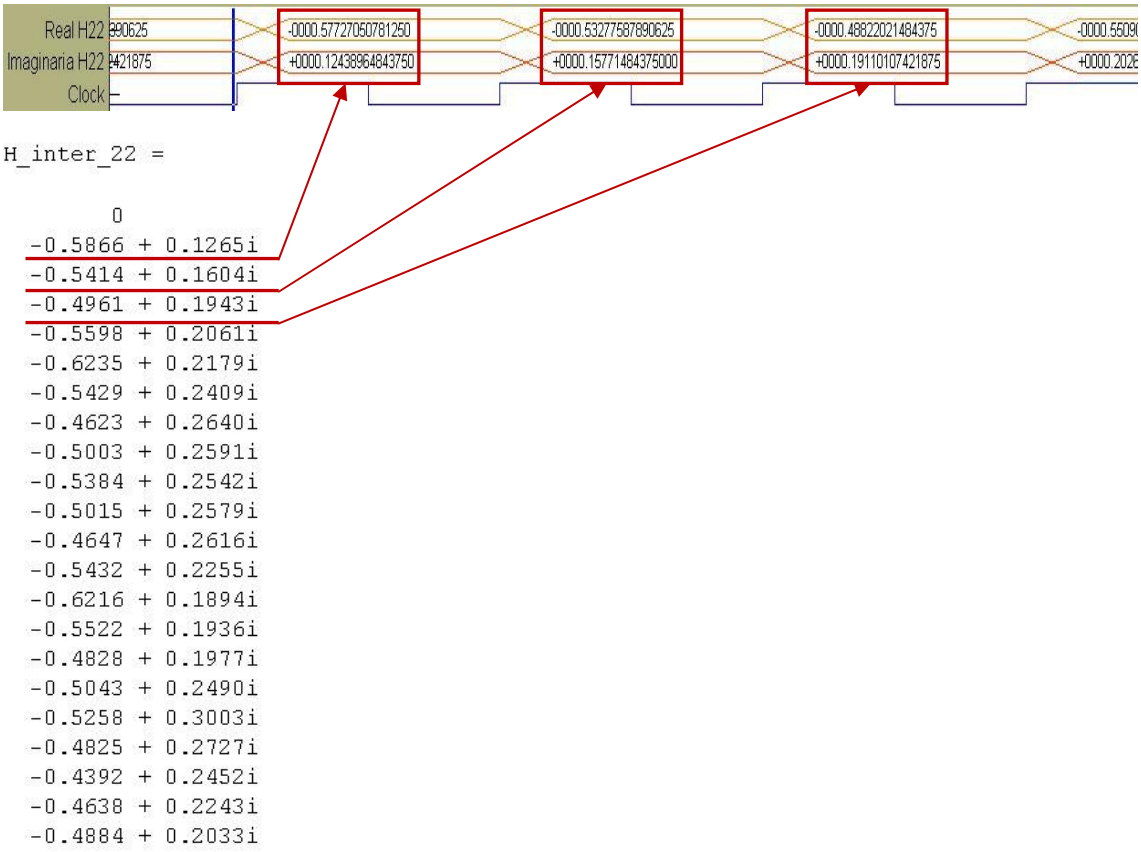
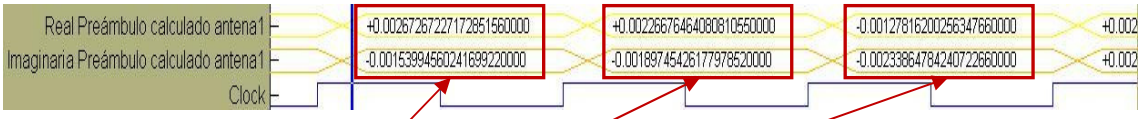


Figura 6.70: Comparativa entre la simulación de Matlab® y del modelo Simulink® para el Canal estimado H22

5.1.2 Salida del bloque "Cálculo del preámbulo recibido suponiendo que la primera estimación de canal es correcta (en el dominio frecuencial)"

En este punto del diseño, vamos a comprobar que el cálculo del preámbulo recibido, tal y como marca el algoritmo, suponiendo que la primera estimación de canal es correcta, se realiza correctamente. Para ello, comparamos los resultados obtenidos de la función de Matlab® en este punto del algoritmo y los resultados obtenidos de la simulación del modelo Simulink®. En la figura (6.71), se ilustra la comparativa entre ambas simulaciones para la rama del diseño referido a la antena 1 (solo se mostrarán las primeras subportadoras con información, omitiendo las bandas de guarda). Por otra parte, en la figura (6.72), se ilustra lo mismo que en la figura (6.68) pero referido a la antena 2.




```
s_recibida_estimada_ant1 =

        0
   0.0027 - 0.0016i
   0.0023 - 0.0019i
  -0.0013 - 0.0024i
   0.0021 + 0.0023i
   0.0025 - 0.0015i
   0.0021 - 0.0019i
  -0.0025 + 0.0014i
   0.0024 - 0.0021i
   0.0025 - 0.0014i
   0.0020 - 0.0018i
   0.0012 + 0.0026i
  -0.0021 - 0.0020i
   0.0015 + 0.0026i
   0.0020 + 0.0017i
  -0.0015 - 0.0026i
   0.0021 + 0.0017i
  -0.0015 - 0.0027i
  -0.0020 - 0.0018i
   0.0026 - 0.0013i
  -0.0017 + 0.0022i
   0.0026 - 0.0016i
   0.0016 - 0.0023i
```

Figura 6.71: Comparativa entre la simulación de Matlab® y del modelo Simulink® para la rama del diseño referida a la antena 1

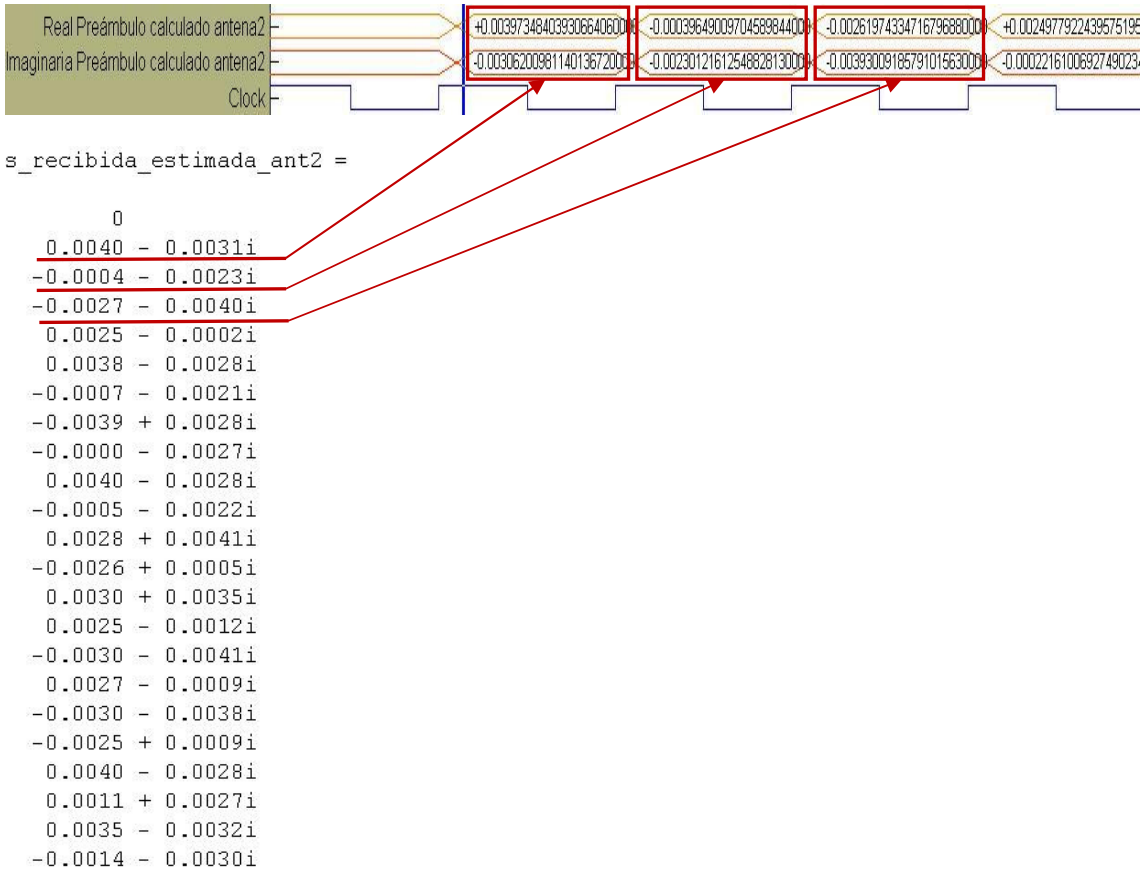
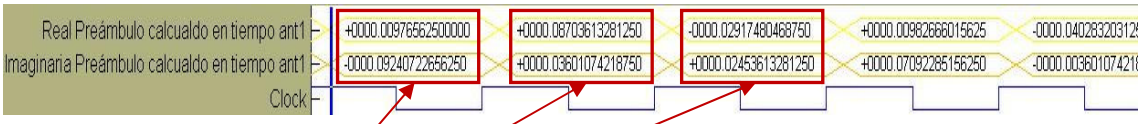


Figura 6.72: Comparativa entre la simulación de Matlab® y del modelo Simulink® para la rama del diseño referida a la antena 2

5.1.3 Salida del bloque "FFT para pasar los preámbulos calculados en cada antena a dominio temporal"

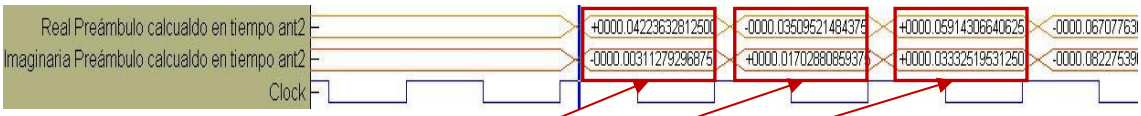
En este punto del diseño, vamos a comprobar que los cálculos realizados en el dominio de la frecuencia en la etapa anterior, han sido transformados correctamente al dominio del tiempo. Para ello, comparamos los resultados obtenidos de la función de Matlab® en este punto del algoritmo y los resultados obtenidos de la simulación del modelo Simulink®. En la figura (6.73), se ilustra la comparativa entre ambas simulaciones para la rama del diseño referido a la antena receptora 1 (solo se mostrarán las primeras muestras). Por otra parte, en la figura (6.74), se ilustra lo mismo que en la figura (6.73) pero referido a la antena receptora 2.




```
s_recibida_estimada_frec_ant1 =

   0.0099 - 0.0938i
   0.0885 + 0.0366i
  -0.0296 + 0.0250i
   0.0100 + 0.0721i
  -0.0409 - 0.0036i
   0.0110 - 0.0634i
  -0.0939 + 0.0845i
   0.0388 - 0.0724i
   0.0922 + 0.0946i
   0.0064 - 0.1315i
   0.0245 - 0.0179i
  -0.0921 + 0.0500i
  -0.0622 + 0.0522i
   0.0534 - 0.0561i
   0.0221 + 0.0918i
   0.1098 - 0.0476i
  -0.1535 - 0.0583i
   0.0631 + 0.1011i
   0.0176 - 0.1499i
```

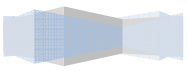
Figura 6.73: Comparativa entre la simulación de Matlab® y del modelo Simulink® para la rama del diseño referida a la antena 1




```
s_recibida_estimada_frec_ant2 =

   0.0429 - 0.0031i
  -0.0356 + 0.0173i
   0.0601 + 0.0339i
  -0.0681 - 0.0836i
   0.0175 + 0.0904i
   0.0603 - 0.0785i
  -0.0714 - 0.0340i
   0.0059 + 0.0744i
  -0.0359 + 0.0149i
   0.0877 + 0.0419i
  -0.0553 - 0.0210i
   0.0062 - 0.0907i
   0.0120 - 0.0449i
   0.0630 + 0.0655i
  -0.0633 + 0.0521i
  -0.0197 + 0.0377i
   0.0052 - 0.0301i
  -0.0251 - 0.0809i
```


Figura 6.74: Comparativa entre la simulación de Matlab® y del modelo Simulink® para la rama del diseño referida a la antena 2



5.1.4 Salida del bloque "Cálculo del "*offset*" frecuencial"

En este punto del diseño, vamos a comprobar que los cálculos realizados para obtener el desplazamiento en frecuencia según la ecuación (6.11), se realizan correctamente. Para ello, comparamos los resultados obtenidos de la función de Matlab® en este punto del algoritmo y los resultados obtenidos de la simulación del modelo Simulink®. En la figura (6.75), se ilustra la comparativa entre ambas simulaciones que nos dan el valor del desplazamiento en frecuencia.

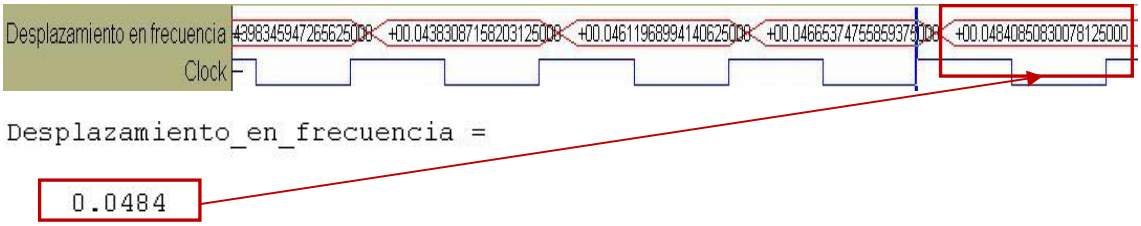
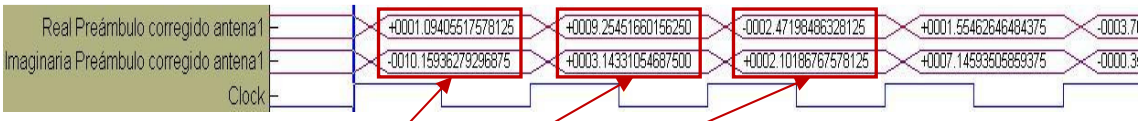


Figura 6.75: Desplazamiento en frecuencia obtenido de la simulación de Matlab® y del modelo Simulink®

Como se puede observar en la figura (6.75), la simulación del modelo Simulink® genera datos de desplazamiento en frecuencia que no son correctos. Esto se debe a la suma recursiva que se debe hacer para modelar el sumatorio que se define en la expresión (6.18). Esta suma recursiva genera valores incorrectos hasta que suma un total de 128 muestras (ver sumatorio de la expresión (6.21)), por tanto, indirectamente, el bloque "***CORDIC ATAN***" generará valores del desplazamiento en frecuencia incorrectos, hasta que se la suma recursiva llega a la muestra número 128.

5.1.5 Salida del bloque "Corrector del "*offset*" frecuencial"

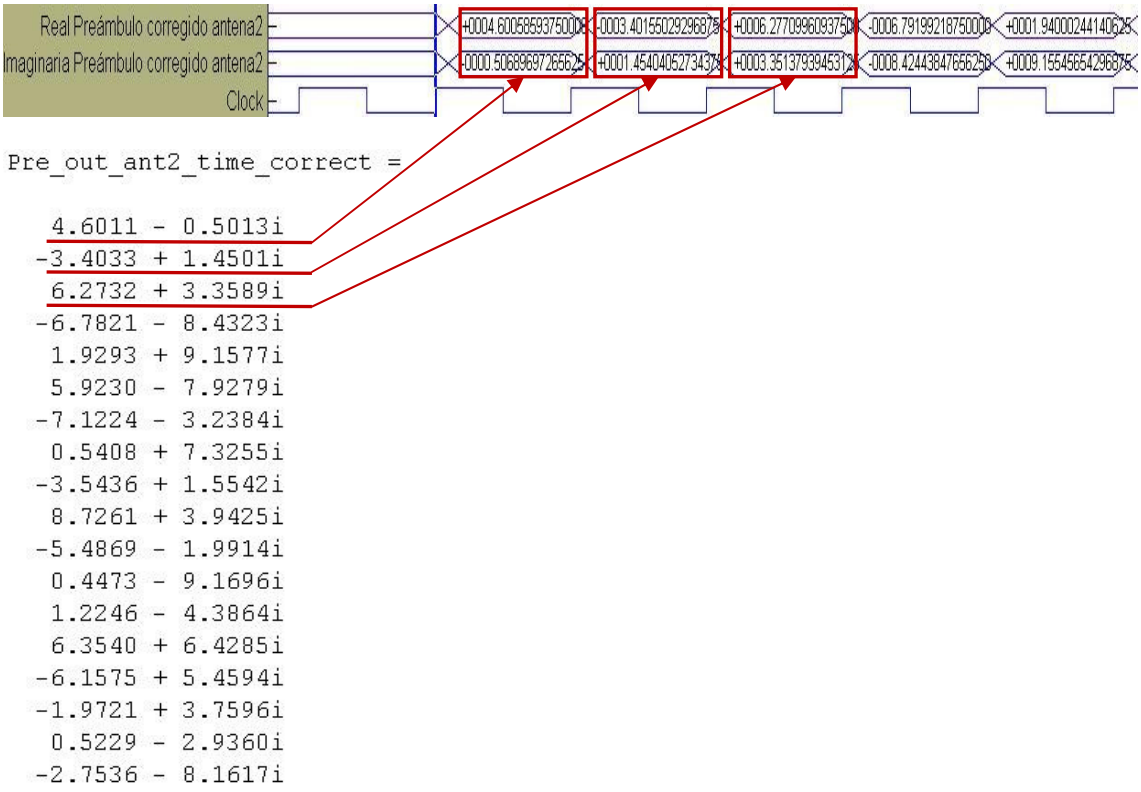
En este punto del diseño, vamos a comprobar que los cálculos realizados para corregir el desplazamiento en frecuencia de los preámbulos recibidos se realiza correctamente. Para ello, comparamos los resultados obtenidos de la función de Matlab® en este punto del algoritmo y los resultados obtenidos de la simulación del modelo Simulink®. En la figura (6.76), se ilustra la comparativa entre ambas simulaciones que nos proporciona el preámbulo corregido para la antena receptora 1 (solo se mostrarán las primeras muestras). Por otra parte, en la figura (6.77), se ilustra lo mismo que en la figura (6.76) pero referido a la antena 2.




```
Pre_out_ant1_time_correct =

   1.1061 -10.1580i
   9.2508 + 3.1543i
  -2.4745 + 2.0990i
   1.5462 + 7.1479i
  -3.7663 - 0.3547i
   1.2066 - 6.1532i
  -9.4718 + 8.5964i
   3.5660 - 7.1366i
   8.9722 + 9.3088i
   0.2951 -13.1644i
   2.3401 - 1.9701i
  -9.0181 + 5.1130i
  -5.9520 + 5.2287i
   5.5148 - 5.6035i
   2.4992 + 9.1493i
  10.9461 - 4.8287i
 -15.5902 - 5.5539i
   6.3388 +10.0374i
```

Figura 6.76: Comparativa entre la simulación de Matlab® y del modelo Simulink® para el preámbulo corregido para la antena 1




```
Pre_out_ant2_time_correct =

   4.6011 - 0.5013i
  -3.4033 + 1.4501i
   6.2732 + 3.3589i
  -6.7821 - 8.4323i
   1.9293 + 9.1577i
   5.9230 - 7.9279i
  -7.1224 - 3.2384i
   0.5408 + 7.3255i
  -3.5436 + 1.5542i
   8.7261 + 3.9425i
  -5.4869 - 1.9914i
   0.4473 - 9.1696i
   1.2246 - 4.3864i
   6.3540 + 6.4285i
  -6.1575 + 5.4594i
  -1.9721 + 3.7596i
   0.5229 - 2.9360i
  -2.7536 - 8.1617i
```

Figura 6.77: Comparativa entre la simulación de Matlab® y del modelo Simulink® para el preámbulo corregido para la antena 2

5.2 Validación del módulo de Sincronización temporal

Para comprobar la validez del hardware diseñado para la sincronización temporal no compararemos los resultados de la simulación de Matlab® con los resultados en de la simulación del modelo Simulink®, como hemos venido mostrando anteriormente. Para validar el hardware, simplemente, ilustraremos las señales de entrada y salida de las principales etapas para comprobar que se realizan los cálculos esperados.

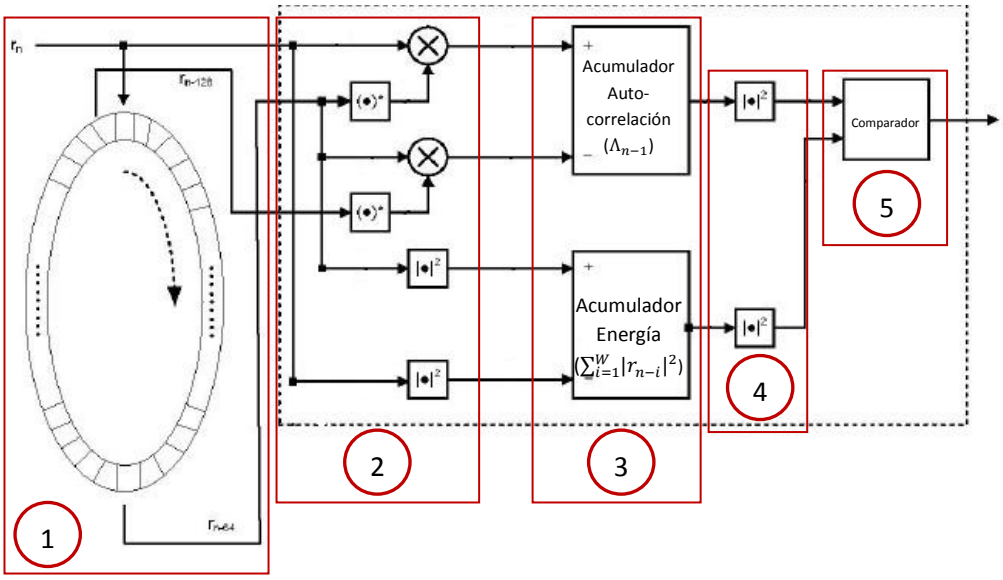


Figura 6.78: Etapas a evaluar

Las etapas a evaluar según la figura (6.78) son:

1) Etapa "*Buffer circular con una capacidad de 128 muestras*"
2) Etapa "*Productos complejos*"
3) Etapa "*Resta y acumulación*"
4) Etapa "*Módulo al cuadrado*"
5) Etapa "*Comparador*"

5.2.1 Etapa "*Buffer* circular con una capacidad de 128 muestras"

En este apartado mostramos la entrada y las salidas de la etapa del diseño que modela al "*buffer*" de la figura (6.79). En la figura (6.79), se muestra la parte real e imaginaria de los datos de entrada al "*buffer*" y la parte real e imaginaria de los datos en las posiciones 64 y 128 del "*buffer*".

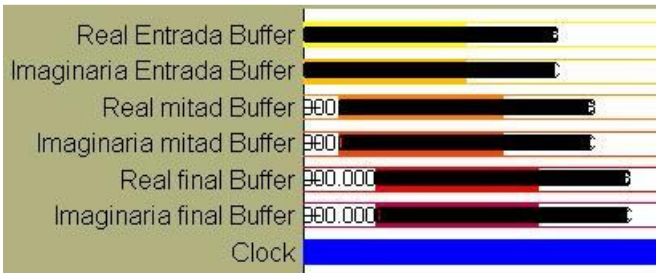


Figura 6.79: Entradas y salidas del buffer

Cuando el preámbulo recibido ha llenado la mitad del "*buffer*", la primera muestra del mismo debe de estar en la posición 64 del "*buffer*". Así, para comprobar esto, no se ha introducido desplazamiento temporal en el preámbulo de entrada al diseño, de manera que la primera muestra del preámbulo debe de aparecer en la posición 64 del "*buffer*", o en otras palabras, en el ciclo de reloj número 64 (64 segundos). También, cuando el "*buffer*" está completamente lleno la primera muestra del preámbulo debe de estar en la posición 128, o en otras palabras, en el ciclo de reloj 128 (128 segundos). En las figuras (6.80), (6.81) y (6.82), se ilustra lo que se acaba de comentar.

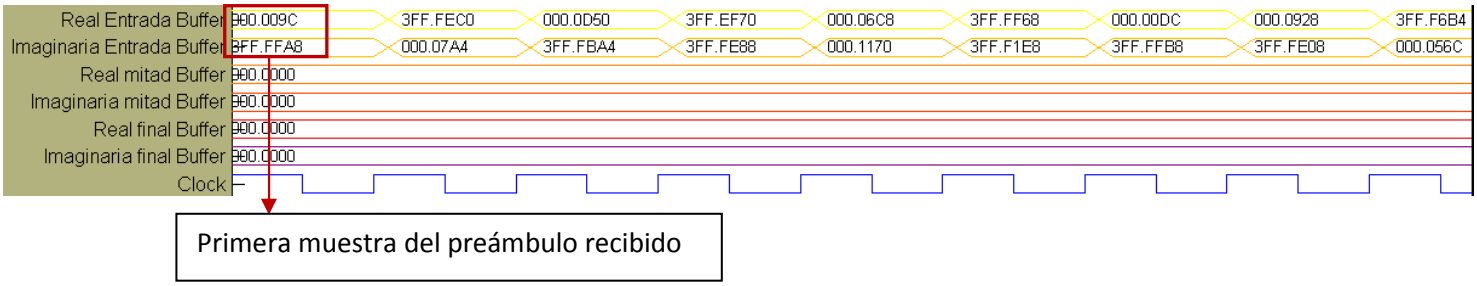


Figura 6.80: Datos de entrada al buffer

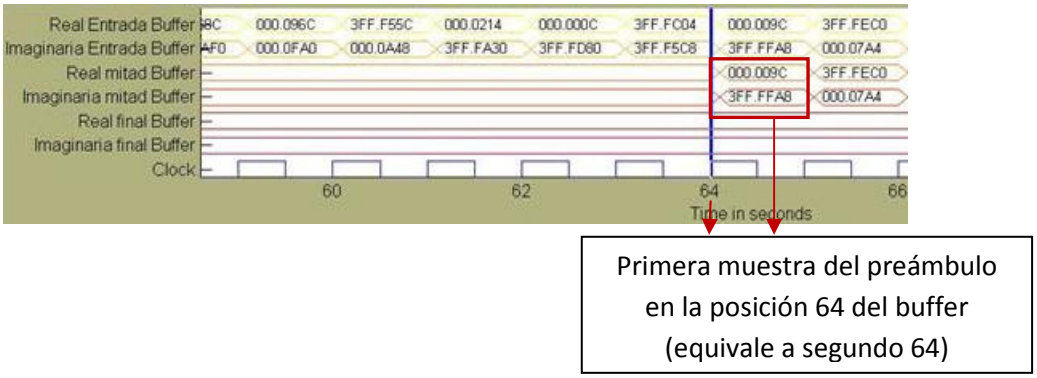


Figura 6.81: Datos de salida del buffer en la posición 64

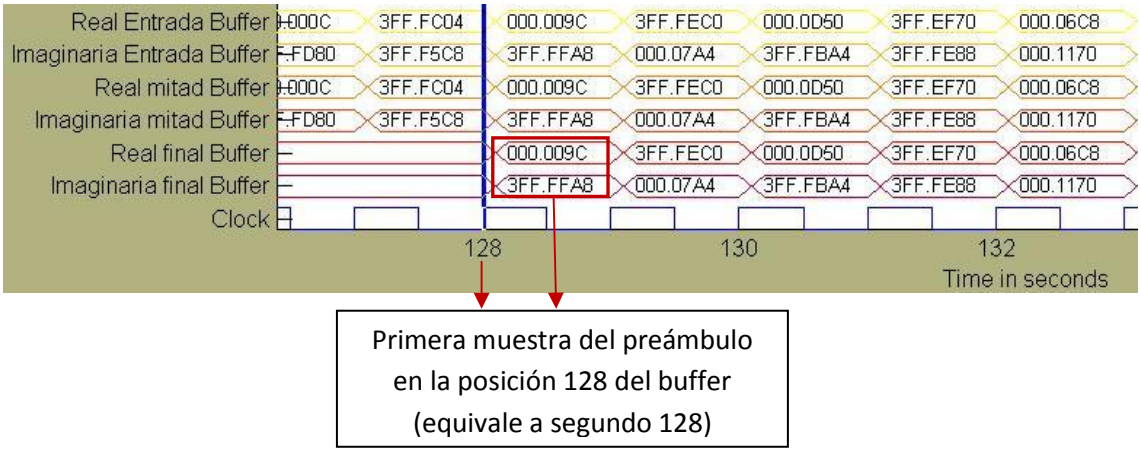


Figura 6.82: Datos de salida del buffer en la posición 128

5.2.2 Etapa "Productos complejos"

En este apartado se ilustrará que los subsistemas que realizan los productos complejos de las expresiones (6.27) y (6.29) realizan tales cálculos correctamente. Únicamente, se mostrarán las entradas y las salidas de uno de los subsistemas *("Producto complejo")*, ya que las operaciones realizadas por el resto siguen los mismos principios. Sobre la figura (6.83), se detallará a modo de ejemplo, como se realiza el producto complejo en un ciclo de reloj.

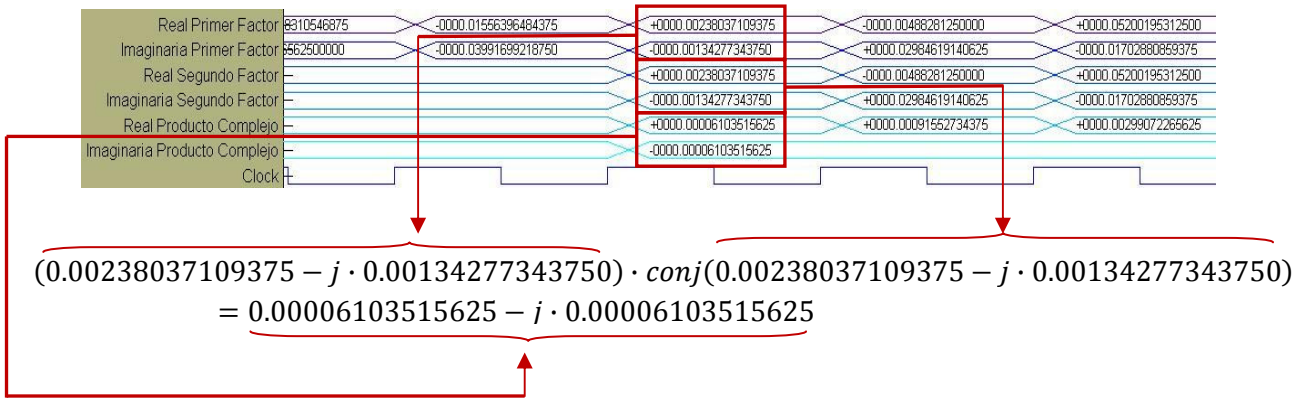


Figura 6.83: Datos de entrada y salida a esta etapa

5.2.3 Etapa "Resta y acumulación"

En este apartado se comprobará que esta etapa del diseño realiza las operaciones de resta y acumulación definidas en las expresiones (6.27) y (6.29). Únicamente, se mostrará las operaciones de resta y acumulación relativas a la ecuación (6.27), ya que la expresión (6.29) sigue los mismos principios. En la figura (6.84) se ilustra cómo se realiza la resta, y en la figura (6.85) cómo se realiza la suma recursiva que implica la acumulación.

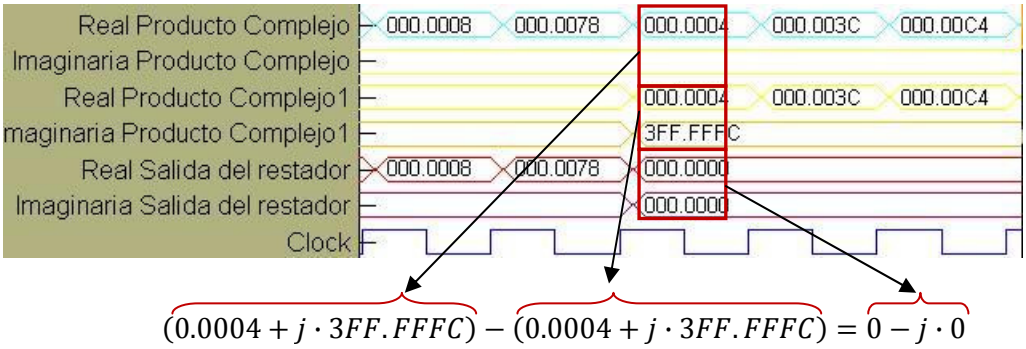


Figura 6.84: Resta de la expresión (6.26)

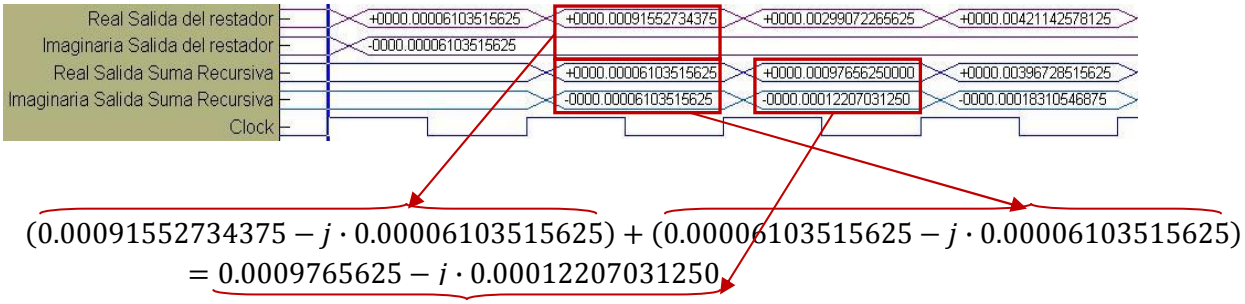


Figura 6.85: Suma recursiva de la expresión (6.26)

Como se deduce de la figura (6.825), la suma recursiva, en un instante de tiempo determinado, suma el resultado de la suma del ciclo anterior con los datos que le llegan en ese instante a su entrada (*Real Salida del restador e Imaginaria Salida del restador*).

5.2.4 Etapa "Módulo al cuadrado"

En este apartado se comprobará que se realiza correctamente la operación de módulo al cuadrado definida tanto en el numerador y denominador de la métrica (6.12). Esta operación, como ya vimos, se realiza mediante un bloque que realiza un producto complejo basándose en el concepto de que el módulo al cuadrado de un número complejo se puede realizar mediante el producto complejo de ese mismo número por su conjugado. Únicamente, se ilustrará el resultado del módulo al cuadrado referido al numerador de la métrica (6.12), ya que para el denominador se opera de la misma manera. Así, en la figura (6.86)

se muestran las entradas y las salidas de esta etapa referidas al numerador de la métrica (6.12).

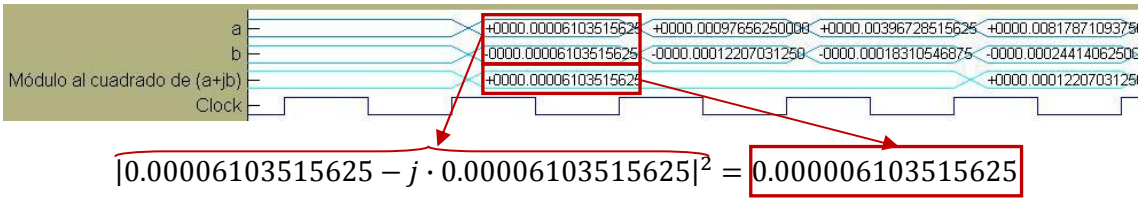


Figura 6.86: Cálculo del módulo al cuadrado

5.2.5 Etapa "Comparador"

En este apartado comprobaremos que el comparador modelado en esta etapa funciona correctamente. En la figura (6.87), se ilustran las entradas y las salidas de este bloque.

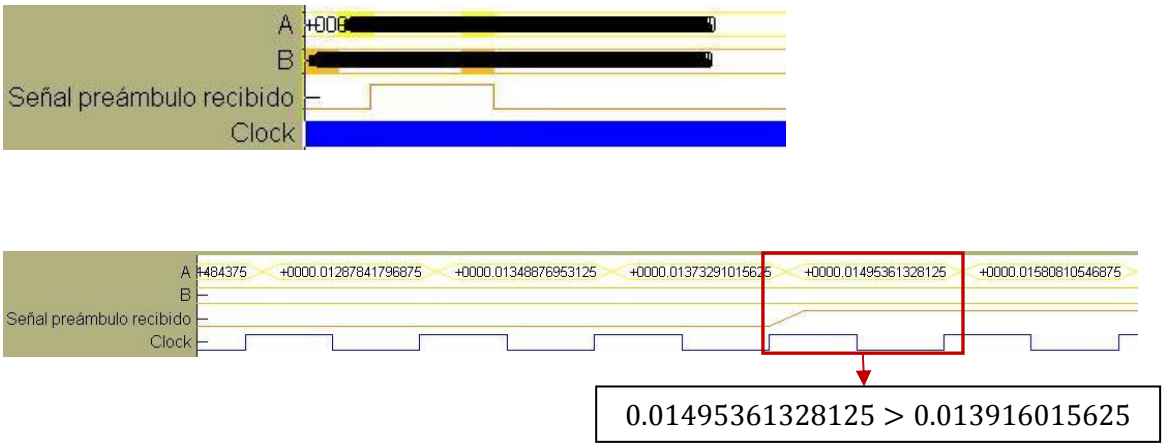


Figura 6.87: Salidas y entradas del bloque comparador

Como podemos ver en la figura (6.87), cuando los datos de la entrada A son mayores a los de la B, la salida del comparador se pone a nivel alto, tal y como se quiere para que se cumpla la condición de la expresión (6.15), la cual recordamos a continuación:

$$\left|\sum_{i=1}^{W} r_{n-i}\cdot r_{n-i-W}^{*}\right|^{2} - th\left|\sum_{i=1}^{W}|r_{n-i}|^{2}\right|^{2} > 0 \rightarrow$$

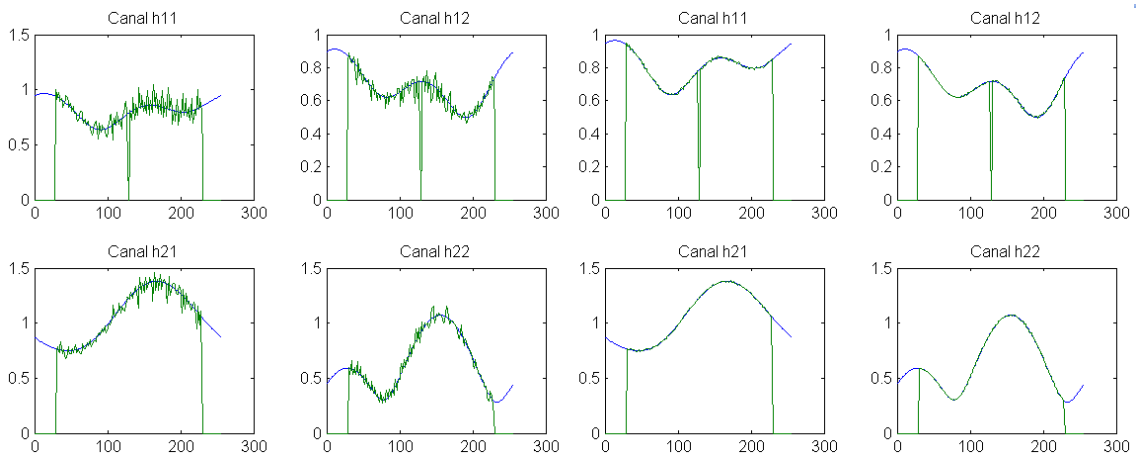
$$\rightarrow \underbrace{\left|\sum_{i=1}^{W} r_{n-i}\cdot r_{n-i-W}^{*}\right|^{2}}_{\mathbf{A}} > \underbrace{th\left|\sum_{i=1}^{W}|r_{n-i}|^{2}\right|^{2}}_{\mathbf{B}} \quad (6.34)$$

6. Resultados

6.1 Módulo de Sincronización en frecuencia

En este apartado mostraremos los resultados del módulo de sincronización en frecuencia. Para ello, importaremos a Matlab® los resultados de la simulación del modelo Simulink® realizado, mediante bloques "***To Workspace***", los cuales generarán vectores en el "*workspace*" de Matlab® que contendrán los resultados de la simulación. Estos vectores serán pasados como entradas a las funciones de Matlab® "*representa.m*" y "*representa3.m*", las cuales serán las encargadas de representar los resultados de la estimación de canal con el desplazamiento frecuencial corregido y no corregido. Estas funciónes de Matlab® la podemos encontrar en el Anexo A.

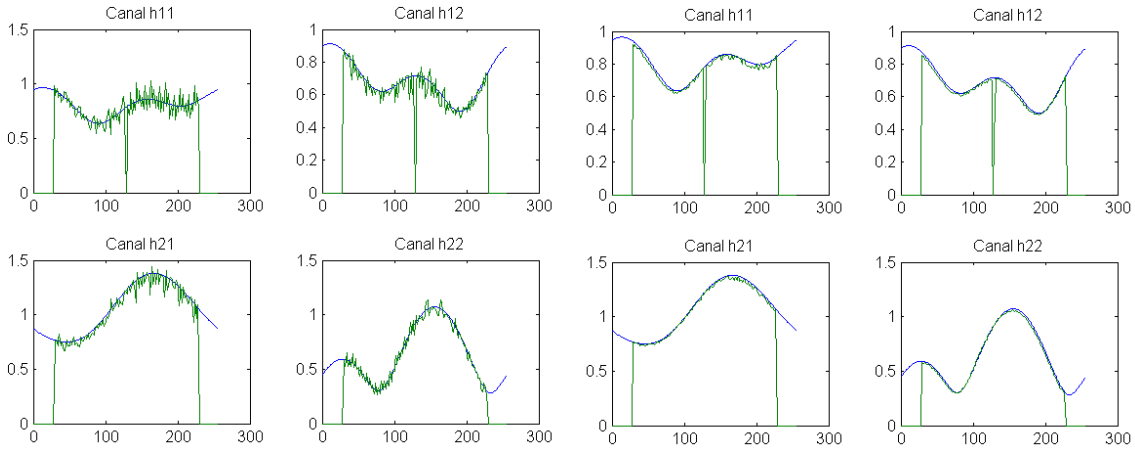
Para comprobar que el resultado final del algoritmo de sincronización en frecuencia es correcto, también, se mostrará el resultado de la simulación de Matlab® (función de Matlab® "*MLTF_offset.m*", ver anexo A) junto con el resultado de la simulación del modelo Simulink®. Así, en la figura (6.88), se muestra el resultado de la simulación de Matlab® del algoritmo de sincronización en frecuencia y, en la figura (6.89), se muestra el resultado de la simulación del modelo Simulink® que modela algoritmo de sincronización en frecuencia. Estas figuras, nos mostrarán como se ha corregido el "*offset*" frecuencial sobre el canal estimado.



a) Primera estimación de Canal

b) Segunda estimación de Canal con el desplazamiento en frecuencia corregido

Figura 6.88: Resultados de la simulación de Matlab®



a) Primera estimación de Canal

b) Segunda estimación de Canal con el desplazamiento en frecuencia corregido

Figura 6.89: Resultados de la simulación del modelo Simulink®

Como podemos ver, tanto para los resultados de Matlab® y como para los de Simulink®, el desplazamiento en frecuencia distorsiona el canal estimado. Esta distorsión es corregida por el algoritmo de sincronización en frecuencia, tal y como es ilustra en las figuras (6.88) y (6.89).

Los resultados obtenidos del modelo Simulink®, son algo menos precisos que los ofrecidos por la simulación de Matlab®. Esto se debe a la limitación en el

número de bits de datos que tiene el hardware diseñado (sólo se emplean datos de 24 bits).

6.2 Módulo de Sincronización temporal

En este apartado mostraremos los resultados del módulo de sincronización temporal. Para ello, simularemos el modelo Simulink® ideado para un preámbulo sin desplazamiento temporal y un preámbulo con cierto desplazamiento temporal. Para ello, generaremos las entradas a nuestro modelo mediante una función de Matlab® ("*Timining.m*") que podemos encontrar en el anexo A. Esta función nos generará el preámbulo transmitido con el desplazamiento que le indiquemos. Este preámbulo generado por la función, será importado del "*Workspace*" de Matlab® al modelo Simulink®, mediante el empleo de bloques "***From Workspace***".

En la figura (6.90), se ilustra la señal de salida del módulo de sincronización temporal para un preámbulo sin desplazamiento temporal, y en la figura (6.91), se ilustra esa misma señal para un preámbulo que tiene un desplazamiento temporal de 50 ciclos de reloj.

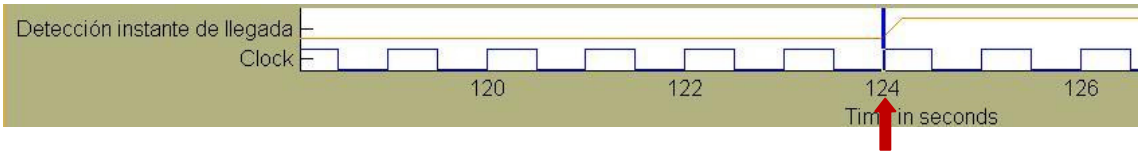


Figura 6.90: Señal de detección del instante de llegada para un desplazamiento de 0 ciclos de reloj

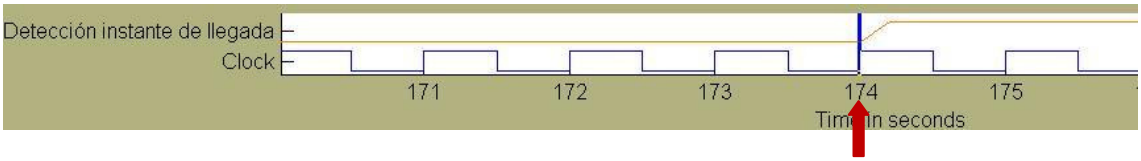


Figura 6.91: Señal de detección del instante de llegada para un desplazamiento de 50 ciclos de reloj

Como se puede deducir de la figura (6.91), el flanco de subida de la señal de salida del módulo de sincronización temporal, el cual determina el instante de llegada del preámbulo, esta retardado 50 (124-50=174) ciclos con respecto al flaco de subida de la señal de la figura (6.90). Este retardo concuerda con el desplazamiento temporal que se le ha aplicado al preámbulo.

Capítulo 7

Líneas de trabajos futuros y Conclusiones

1. Líneas de trabajos futuros

Los trabajos futuros a realizar están orientados a la integración de los módulos diseñados en un sistema de comunicaciones (*emisor y receptor*). Así, los módulos de estimación de canal y sincronización tiempo-frecuencia diseñados en el presente proyecto se idearon para ser integrados en el sistema de comunicaciones ideado por los autores David Díaz Martín [36] y Roberto Prieto Alonso. Para ello, es necesaria una amplia modificación de tal sistema de comunicaciones. Las principales modificaciones que deberían de realizarse son las siguientes:

1) En el emisor del sistema debería de diseñarse un generador del preámbulo propuesto en el presente proyecto para comprobar que la inserción del los módulos diseñados funcionan correctamente.

2) Es necesaria la integración de un bloque en el modelo Simulink® del sistema de comunicaciones que simule el canal sin necesidad de utilizar bloques "***From Workspace***".

3) Es necesario rediseñar el extractor del preámbulo, ya que el extractor actualmente diseñado simplemente elimina el preámbulo en lugar de extraerlo. Al no tener en cuenta la estimación de canal y la sincronización tiempo-frecuencia, no se requirió el empleo del preámbulo, por lo que simplemente se eliminó. Por tanto, es necesario diseñar otro módulo extractor del preámbulo que no lo elimine, ya que será necesaria su utilización.

4) El diseño del sistema de comunicaciones utiliza bloques "**FFT v1_0**", los cuales no son sintetizables sobre la FPGA que se ha utilizado, una Virtex-4. Por tanto, es necesario cambiar todos los bloques "**FFT v1_0"** del sistema de comunicaciones por otros que sean sintetizables. Para ello, se propone utilizar el bloque "**FFT v3_1**", el cual ha sido empleado en los módulos diseñados en el presente proyecto.
5) El sincronismo del sistema de comunicaciones diseñado emplea bloques "***Delay***". Esto no es un buen diseño, ya que se está aumentando en exceso el área del hardware diseñado. Por lo tanto, se propone rediseñar todo el sincronismo del sistema, de manera que se utilicen señales de control de datos en lugar bloques "***Delay***". Así, se reduciría el área del hardware diseñado.

6) El número de bits de datos empleados en el sistema de comunicaciones es menor que el empleado para el diseño de los módulos del presente proyecto. Así, para integrar los módulos diseñados se requerirá aumentar el número de bits de datos empleados en el sistema de comunicaciones, con el fin de no perder calidad y precisión de las estimaciones realizadas.

2. Conclusiones

De la realización del presente proyecto se extraen conclusiones muy positivas. Las principales conclusiones extraídas del mismo son:

1) Se han estudiado los fundamentos básicos de los sistemas MIMO-OFDM.
2) Se han estudiado los fundamentos básicos de una nueva herramienta de diseño hardware que no se había utilizado antes (plataforma de desarrollo para FPGA basada en Simulink® *VHS-ADC Virtex-4 de Lyrtech® junto con la herramienta Xilinx System Generator For DSP®).*
3) Se ha consultado cierta bibliografía relativa a algoritmos de sincronización tiempo-frecuencia y estimación de canal que se desconocían.
4) Se ha logrado realizar el diseño de los módulos de estimación de canal y sincronización tiempo-frecuencia para un sistema de comunicaciones basado en el estándar 802.16d-2004 (*Wimax Fijo*) utilizando MIMO 2x2, OFDM y BPSK para la modulación de datos.

Capítulo 8

Presupuesto

En este capítulo se detalla el coste de realización del presente proyecto final de carrera. Así, en los siguientes apartados se detallará el coste de recursos, tanto materiales como personales, para finalizar el capítulo con el detalle del coste total del proyecto.

1. Coste personal

En este apartado se detalla el coste de la mano de obra implicada en la realización del proyecto. Para calcular este coste será preciso detallar las horas empleadas en cada una de las fases seguidas para la realización del proyecto.

En una primera fase, se adquirieron todos los conocimientos necesarios para llevar a cabo el proyecto mediante el estudio teórico de distintas fuentes de información. En una segunda fase, se llevaron a la práctica los conocimientos adquiridos en la primera fase (descripción del hardware, simulaciones realizadas, obtención e interpretación de resultados). Por último, en una tercera fase, se redactó la presente memoria.

Así, en la siguiente tabla (7.1), se detallan el número de horas empleadas en cada una de las fases:

Fase	Concepto	Horas
1	Documentación y estudio teórico	420 (en 4 meses)
2	Descripción hardware en VHDL	360 (en 3 meses)
3	Redacción de memoria	270 (en 2 meses)

Tabla 7.1: Número de horas empleadas en la realización del PFC

El total de horas trabajadas son 1050 horas. A esta cantidad de horas, hay que sumarle el número de horas dedicadas por los tutores del proyecto a la supervisión del mismo. Así, estimaremos un total de 100 horas para la

supervisión del proyecto por parte de los tutores. Para calcular el coste asociado a la mano de obra, nos basamos en la tabla de honorarios del *colegio oficial de ingenieros de telecomunicación* [5], el cual fija un coste de 70 €/hora trabajada. Por tanto, se detalla en la siguiente tabla el coste de personal total:

Concepto	**Tarifa**	**Horas**	**Coste (€)**
Proyectante	70 €/hora	1050	73500
Tutores	70 €/hora	100	7000
TOTAL (Coste personal)			**80500**

Tabla 7.2: Coste personal

2. Coste de material

El coste de material se detalla en la tabla (7.3). En la tabla (7.3) es de destacar ciertos aspectos de ella. Cabe destacar que la plataforma de desarrollo VHS-ADC Virtex-4, la herramienta Matlab®/Simulink® y la herramienta Xilinx System Generator®, están compartidas por el departamento de teoría de la señal, por lo que los costes que parecen en la tabla (7.3) hacen referencia al coste imputable para este proyecto.

Concepto	**Coste Real (€)**	**Coste imputable (€)**
Ordenador portátil con Windows XP	700	700
VHS-ADC Virtex-4 de Lyrtech® con Windows XP	18000	3000
Matlab® y Simulink®	3000	100
Xilinx System Generator for DSP®	6000	200
Conexión internet	40	40
Impresión de documentos	120	120
TOTAL (Coste material)		**4160**

Tabla 7.3: Coste de material

3. Costes Totales

Los costes totales implican la suma de los costes de personal y de material. Estos costes son detallados en la tabla (7.4):

Concepto	**Coste (€)**
Coste personal	80500
Coste de material	4160
TOTAL	**84660**

Tabla 7.4: Costes Totales

Anexo A

El objetivo de este anexo es proporcionar los códigos de Matlab® utilizados en el presente proyecto.

1) MLTF_offset.m

```
function [H_11_re H_12_re H_21_re H_22_re offset En_re_Pre_out_ant1_time En_im_Pre_out_ant1_time
En_re_Pre_out_ant2_time En_im_Pre_out_ant2_time]=MLTF_offset(epsilon)

p_all = [1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, ...
   -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, ...
   1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, ...
   -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, ...
   1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, ...
   1+j, 1+j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, 1+j, 1+j, -1+j, 1-j, ...
   -1+j, -1+j, 1-j, -1+j, 1-j, 1-j, 1+j, -1-j, -1-j, -1-j, -1+j, 1-j, ...
   -1-j, -1-j, 1+j, -1-j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1+j, 1-j, ...
   -1+j, -1+j, -1-j, 1+j, 0, -1-j, 1+j, -1+j, -1+j, -1-j, 1+j, 1+j, ...
   1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j, -1+j, -1+j, 1-j, -1-j, ...
   -1-j, -1+j, 1-j, 1+j, 1+j, -1+j, 1-j, 1-j, 1-j, -1+j, 1-j, -1-j, ...
   -1-j, -1-j, 1+j, 1+j, 1+j, 1+j, -1-j, -1+j, -1+j, 1+j, -1-j, 1-j, ...
   1-j, 1+j, -1-j, -1-j, -1-j, 1+j, -1-j, -1+j, -1+j, -1+j, 1-j, 1-j, ...
   1-j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, -1+j, -1+j, -1-j, 1+j, 1+j, ...
   1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j, -1+j, -1+j, 1-j, -1-j, ...
   -1-j, 1-j, -1+j, -1-j, -1-j, 1-j, -1+j, -1+j, -1+j, 1-j, -1+j, 1+j, ...
   1+j, 1+j, -1-j, -1-j, -1-j, -1-j, 1+j, 1-j, 1-j].';

p_even = zeros(length(p_all),1);
p_odd = zeros(length(p_all),1);
% Preámbulo P_EVEN.
p_even(1:2:end) = sqrt(2)*p_all(1:2:end);
% Preámbulo P_ODD.
p_odd(2:2:end) = sqrt(2)*p_all(2:2:end);
% Preámbulo P_EVEN con bandas de guarda.
p_even_c = [complex(zeros(28,1));p_even;complex(zeros(27,1))];
% Preámbulo P_ODD con bandas de guarda.
p_odd_c = [complex(zeros(28,1));p_odd;complex(zeros(27,1))];

Pre=[p_even_c.';p_odd_c.'];
R=Pre'*inv(Pre*Pre');

% simulación del canal

coef_chan=zeros(4,3);

for i=1:4
   if i==1
      coef_chan(i,:)=chan_SUI(1,256,[0 -15 -20],[4 0 0],[0.4e-6 0.9e-6],[0.4 0.3 0.5],-0.1771); % h11
   elseif i==2
      coef_chan(i,:)=chan_SUI(1,256,[0 -12 -15],[2 0 0],[0 0.4e-6 1.1e-6],[0.2 0.15 0.25],-0.3930); %h12
   elseif i==3
      coef_chan(i,:)=chan_SUI(1,256,[0 -5 -10],[1 0 0],[0 0.5e-6 1e-6],[0.4 0.3 0.5],-1.5113); %h21
   else
      coef_chan(i,:)=chan_SUI(1,256,[0 -4 -8],[0 0 0],[0 1.5e-6 4e-6],[0.2 0.15 0.25],-1.9218); %h22
```

```
    end
end

%respuesta en frecuencia de los 4 canales

H_11_re=fft(coef_chan(1,:),256);
H_12_re=fft(coef_chan(2,:),256);
H_21_re=fft(coef_chan(3,:),256);
H_22_re=fft(coef_chan(4,:),256);

%preambulo transmitido con una desviación de frecuencia epsilon

c=[0:255];
d=exp(j*2*pi*epsilon*c/256);

p_even_c_despf=fft(ifft(p_even_c,256).*d.',256); %cambio a tiempo, multiplico por las exponenciales y vuelvo a
frecuencia
p_odd_c_despf=fft(ifft(p_odd_c,256).*d.',256);

%respuesta en frecuencia de los preambulos a la salida de cada canal

p_even_c_despf_out_11=H_11_re.*p_even_c_despf.';
p_even_c_despf_out_12=H_12_re.*p_even_c_despf.';
p_odd_c_despf_out_21=H_21_re.*p_odd_c_despf.';
p_odd_c_despf_out_22=H_22_re.*p_odd_c_despf.';

%Preambulos recibidos en cada antena. Los multiplicamos por cien debido a
%la atenuación del canal

Pre_out_ant1=(p_even_c_despf_out_11.'+p_odd_c_despf_out_21.')*100;
Pre_out_ant2=(p_even_c_despf_out_12.'+p_odd_c_despf_out_22.')*100;

Pre_out=[Pre_out_ant1 Pre_out_ant2].';

%coeficientes de canal estimados en frecuencia para cada uno de los 4
%canales

H_11=zeros(1,256);
H_12=zeros(1,256);
H_21=zeros(1,256);
H_22=zeros(1,256);

for i=1:256
    H=Pre_out(:,i)*R(i,:);
    H_11(i)=H(1,1);
    H_21(i)=H(2,1);
    H_12(i)=H(1,2);
    H_22(i)=H(2,2);
end

H_inter_11=zeros(1,256);
H_inter_12=zeros(1,256);
H_inter_21=zeros(1,256);
H_inter_22=zeros(1,256);

for i=29:2:229
    H_inter_11(i+1)=(H_11(i)+H_11(i+2))/2;
```

```
   H_inter_21(i+1)=(H_21(i)+H_21(i+2))/2;
end

for i=30:2:230
   H_inter_22(i+1)=(H_22(i)+H_22(i+2))/2;
   H_inter_12(i+1)=(H_12(i)+H_12(i+2))/2;
end

H_inter_11=H_inter_11+H_11;
H_inter_12=H_inter_12+H_12;
H_inter_21=H_inter_21+H_21;
H_inter_22=H_inter_22+H_22;

s_recibida_ant1=H_inter_11.'.*p_even_c+H_inter_12.'.*p_odd_c;
s_recibida_ant2=H_inter_21.'.*p_even_c+H_inter_22.'.*p_odd_c;

%multiplicamos por 100 despues de la ifft para amplificar la señal recibida
%y porque es mas sencillo hacer la ifft antes y luego multiplicar

s_recibida_ant1_time=(ifft(s_recibida_ant1,256))*100;
s_recibida_ant2_time=(ifft(s_recibida_ant2,256))*100;

Pre_out_ant1_time=ifft(Pre_out_ant1,256);
Pre_out_ant2_time=ifft(Pre_out_ant2,256);

offset1=sum(conj(Pre_out_ant1_time(1:128).*conj(s_recibida_ant1_time(1:128))).*(Pre_out_ant1_time(129:256).*
conj(s_recibida_ant1_time(129:256))));
offset2=sum(conj(Pre_out_ant2_time(1:128).*conj(s_recibida_ant2_time(1:128))).*(Pre_out_ant2_time(129:256).*
conj(s_recibida_ant2_time(129:256))));
offset=10*angle(offset1+offset2);

%definimos las señales que entrarán a nuestro modelo simulink
n=0:255;

En_re_Pre_out_ant1_time=[n.' real(Pre_out_ant1_time)];
En_im_Pre_out_ant1_time=[n.' imag(Pre_out_ant1_time)];
En_re_Pre_out_ant2_time=[n.' real(Pre_out_ant2_time)];
En_im_Pre_out_ant2_time=[n.' imag(Pre_out_ant2_time)];

%compensamos el desplazamiento en frecuencia

desp=exp(j*2*pi*(-offset)*c/256);

Pre_out_ant1_time_correct=Pre_out_ant1_time.*desp.';
Pre_out_ant2_time_correct=Pre_out_ant2_time.*desp.';

%paso el preámbulo compensado al dominio de la frecuencia ya que la
%estimación de canal se realiza en frecuencia

Pre_out_ant1_correct=fft(Pre_out_ant1_time_correct, 256);
Pre_out_ant2_correct=fft(Pre_out_ant2_time_correct, 256);

Pre_out_correct=[Pre_out_ant1_correct Pre_out_ant2_correct].';

%coeficientes de canal estimados en frecuencia para cada uno de los 4
%canales con el offset compensado
```

```
H_correct_11=zeros(1,256);
H_correct_12=zeros(1,256);
H_correct_21=zeros(1,256);
H_correct_22=zeros(1,256);

for i=1:256
   H_correct=Pre_out_correct(:,i)*R(i,:);
   H_correct_11(i)=H_correct(1,1);
   H_correct_21(i)=H_correct(2,1);
   H_correct_12(i)=H_correct(1,2);
   H_correct_22(i)=H_correct(2,2);
end

H_inter_correct_11=zeros(1,256);
H_inter_correct_12=zeros(1,256);
H_inter_correct_21=zeros(1,256);
H_inter_correct_22=zeros(1,256);

for i=29:2:229
   H_inter_correct_11(i+1)=(H_correct_11(i)+H_correct_11(i+2))/2;
   H_inter_correct_21(i+1)=(H_correct_21(i)+H_correct_21(i+2))/2;
end

for i=30:2:230
   H_inter_correct_22(i+1)=(H_correct_22(i)+H_correct_22(i+2))/2;
   H_inter_correct_12(i+1)=(H_correct_12(i)+H_correct_12(i+2))/2;
end

H_inter_correct_11=H_inter_correct_11+H_correct_11;
H_inter_correct_12=H_inter_correct_12+H_correct_12;
H_inter_correct_21=H_inter_correct_21+H_correct_21;
H_inter_correct_22=H_inter_correct_22+H_correct_22;

x=1:256;

subplot (2,2,1)
plot(x,abs(H_11_re),x,abs(H_inter_11))
title('Canal h11')
subplot (2,2,2)
plot(x,abs(H_12_re),x,abs(H_inter_21))
title('Canal h12')
subplot (2,2,3)
plot(x,abs(H_21_re),x,abs(H_inter_12))
title('Canal h21')
subplot (2,2,4)
plot(x,abs(H_22_re),x,abs(H_inter_22))
title('Canal h22')

figure

subplot (2,2,1)
plot(x,abs(H_11_re),x,abs(H_inter_correct_11))
title('Canal h11')
subplot (2,2,2)
plot(x,abs(H_12_re),x,abs(H_inter_correct_21))
title('Canal h12')
subplot (2,2,3)
```

```
plot(x,abs(H_21_re),x,abs(H_inter_correct_12))
title('Canal h21')
subplot (2,2,4)
plot(x,abs(H_22_re),x,abs(H_inter_correct_22))
title('Canal h22')
```

2) Timing.m

```
function [En_re_desplazado1 En_im_desplazado1]=Timing(desplazamiento)

p_all = [1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, ...
   -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, ...
   1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, ...
   -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, ...
   1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, ...
   1+j, 1+j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, 1+j, 1+j, -1+j, 1-j, ...
   -1+j, -1+j, 1-j, -1+j, 1-j, 1-j, 1+j, -1-j, -1-j, -1-j, -1+j, 1-j, ...
   -1-j, -1-j, 1+j, -1-j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1+j, 1-j, ...
   -1+j, -1+j, -1-j, 1+j, 0, -1-j, 1+j, -1+j, -1+j, -1-j, 1+j, 1+j, ...
   1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j, -1+j, -1+j, 1-j, -1-j, ...
   -1-j, -1+j, 1-j, 1+j, 1+j, -1+j, 1-j, 1-j, 1-j, -1+j, 1-j, -1-j, ...
   -1-j, -1-j, 1+j, 1+j, 1+j, 1+j, -1-j, -1+j, -1+j, 1+j, -1-j, 1-j, ...
   1-j, 1+j, -1-j, -1-j, -1-j, 1+j, -1-j, -1+j, -1+j, -1+j, 1-j, 1-j, ...
   1-j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, -1+j, -1+j, -1-j, 1+j, 1+j, ...
   1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j, -1+j, -1+j, 1-j, -1-j, ...
   -1-j, 1-j, -1+j, -1-j, -1-j, 1-j, -1+j, -1+j, -1+j, 1-j, -1+j, 1+j, ...
   1+j, 1+j, -1-j, -1-j, -1-j, -1-j, 1+j, 1-j, 1-j].';
p_even = zeros(length(p_all),1);
p_odd = zeros(length(p_all),1);
p_4x64 = zeros(length(p_all),1);
% Preámbulo P_EVEN.
p_even(1:2:end) = sqrt(2)*p_all(1:2:end);
% Preámbulo P_ODD.
p_odd(2:2:end) = sqrt(2)*p_all(2:2:end);
%Preámbulo P_4x64
p_4x64(1:4:end)=sqrt(2)*sqrt(2)*conj(p_all(1:4:end));
% Preámbulo P_EVEN con bandas de guarda.
p_even_c = [complex(zeros(28,1));p_even;complex(zeros(27,1))];
% Preámbulo P_ODD con bandas de guarda.
p_odd_c = [complex(zeros(28,1));p_odd;complex(zeros(27,1))];
% Preámbulo P_4x64 con bandas de guarda.
p_4x64_c = [complex(zeros(28,1));p_4x64;complex(zeros(27,1))];


% simulación del canal

coef_chan=zeros(4,3);

for i=1:4
   if i==1
      coef_chan(i,:)=chan_SUI(1,256,[0 -15 -20],[4 0 0],[0.4e-6 0.9e-6],[0.4 0.3 0.5],-0.1771); % h11
   elseif i==2
      coef_chan(i,:)=chan_SUI(1,256,[0 -12 -15],[2 0 0],[0 0.4e-6 1.1e-6],[0.2 0.15 0.25],-0.3930); %h12
   elseif i==3
      coef_chan(i,:)=chan_SUI(1,256,[0 -5 -10],[1 0 0],[0 0.5e-6 1e-6],[0.4 0.3 0.5],-1.5113); %h21
   else
```

```
      coef_chan(i,:)=chan_SUI(1,256,[0 -4 -8],[0 0 0],[0 1.5e-6 4e-6],[0.2 0.15 0.25],-1.9218); %h22
    end
end

%respuesta en frecuencia de los 4 canales

H_11_re=fft(coef_chan(1,:),256);
H_12_re=fft(coef_chan(2,:),256);
H_21_re=fft(coef_chan(3,:),256);
H_22_re=fft(coef_chan(4,:),256);

%preambulo transmitido con una desviación de frecuencia epsilon
epsilon=0;

c=[0:255];
d=exp(j*2*pi*epsilon*c/256);

p_even_c_despf=fft(ifft(p_even_c,256).*d.',256); %cambio a tiempo, multiplico por las exponenciales y vuelvo a
frecuencia
p_odd_c_despf=fft(ifft(p_odd_c,256).*d.',256);

%respuesta en frecuencia de los preambulos a la salida de cada canal

p_even_c_despf_out_11=H_11_re.*p_even_c_despf.';
p_even_c_despf_out_11_ivn=ifft(p_even_c_despf_out_11,256).'*100;

p_4x64_c_out_11=H_11_re.*p_4x64_c.';
p_4x64_c_out_11_ivn=ifft(p_4x64_c_out_11,256).'*100;

p_even_c_despf_out_12=H_12_re.*p_even_c_despf.';
p_odd_c_despf_out_21=H_21_re.*p_odd_c_despf.';
p_odd_c_despf_out_21_ivn=ifft(p_odd_c_despf_out_21,256).'*100;

p_odd_c_despf_out_22=H_22_re.*p_odd_c_despf.';

%Preambulos recibidos en cada antena. Los multiplicamos por cien debido a
%la atenuación del canal

Pre_out_ant1=(p_even_c_despf_out_11.'+p_odd_c_despf_out_21.')*100;
Pre_out_ant2=(p_even_c_despf_out_12.'+p_odd_c_despf_out_22.')*100;

Pre_out_ant1_time=ifft(Pre_out_ant1,256);
Pre_out_ant2_time=ifft(Pre_out_ant2,256);

%vemos que si añado la otra parte del preambulo los resultados emperoran,
%por lo que es preferible usar solo un preambulo. También notar que hay un
%momento que se da la indeterminación 0/0 por lo que matlab probablemente
%muestre un valor constante aleatorio como consecuencia de la
%indeterminacion. Dependiendo de la ejecución ese valor varia. Esto puede
%esplicar el plateau tras cierto instante de tiempo, que para
%desplazamiento=1 suele situarse en el instante 415-416.
%Pre_out_ant1_time=[p_even_c_despf_out_11_ivn;
p_even_c_despf_out_11_ivn(1:32)];%+[p_odd_c_despf_out_21_ivn; p_odd_c_despf_out_21_ivn(1:32)];
Pre_out_ant1_time=[p_4x64_c_out_11_ivn(225:256); p_4x64_c_out_11_ivn];

%calculo del desplazamiento temporal
```

```
desplazado1=zeros(2048,1);
desplazado2=zeros(2048,1);

%desplazado1(desplazamiento:(255+desplazamiento))=Pre_out_ant1_time;
desplazado1(desplazamiento:(287+desplazamiento))=Pre_out_ant1_time;

g=0:2047;
En_re_desplazado1=[g.' real(desplazado1)];
En_im_desplazado1=[g.' imag(desplazado1)];

desplazado1=desplazado1/64;

buffer=zeros(128,1);
P=zeros(2048,1);
R=zeros(2048,1);
acumulador1=0;
acumulador2=0;

for i=1:2048
   P(i)=acumulador1+desplazado1(i).*conj(buffer(64))-buffer(64).*conj(buffer(128));
   w(i)=desplazado1(i).*conj(buffer(64));
   b(i)=buffer(64).*conj(buffer(128));
   u(i)=w(i)-b(i);
   acumulador1=P(i);
   R(i)=acumulador2+(abs(buffer(64))^2)-(abs(desplazado1(i))^2);
   acumulador2=R(i);

   M(i)=(abs(P(i)).^2)./(R(i).^2);

   buffer=[0; buffer(1:127)];%desplazo buffer
   buffer(1)=desplazado1(i);%actualizo primera posición del buffer.
end

[valor indice]=max(P);

[w.' b.' u.'];

x=1:2048;

subplot (2,2,1)
plot(R.^2)
title('R.^2')
subplot (2,2,2)
plot(abs(P).^2)
title('abs(P).^2')
subplot (2,2,3)
plot(M)
title('M')
subplot (2,2,4)
plot(x,R.^2,x,abs(P).^2)
title('R.^2 & abs(P).^2')
```

3) Representa.m

```
function
[]=representa(H_11_re,H_12_re,H_21_re,H_22_re,H_11_real_ML,H_11_imag_ML,H_12_real_ML,H_12_imag_ML,H
_21_real_ML,H_21_imag_ML,H_22_real_ML,H_22_imag_ML)

H_11_real=[zeros(28,1); H_11_real_ML(616:816); zeros(27,1)];
H_11_imag=[zeros(28,1); H_11_imag_ML(616:816); zeros(27,1)];
H_12_real=[zeros(28,1); H_12_real_ML(616:816); zeros(27,1)];
H_12_imag=[zeros(28,1); H_12_imag_ML(616:816); zeros(27,1)];
H_21_real=[zeros(28,1); H_21_real_ML(616:816); zeros(27,1)];
H_21_imag=[zeros(28,1); H_21_imag_ML(616:816); zeros(27,1)];
H_22_real=[zeros(28,1); H_22_real_ML(616:816); zeros(27,1)];
H_22_imag=[zeros(28,1); H_22_imag_ML(616:816); zeros(27,1)];

H_11_esti=H_11_real+H_11_imag*j;
H_12_esti=H_12_real+H_12_imag*j;
H_21_esti=H_21_real+H_21_imag*j;
H_22_esti=H_22_real+H_22_imag*j;

x=1:256;
subplot (2,2,1)
plot(x,abs(H_11_re),x,abs(H_11_esti))
title('Canal h11')
subplot (2,2,2)
plot(x,abs(H_12_re),x,abs(H_12_esti))
title('Canal h12')
subplot (2,2,3)
plot(x,abs(H_21_re),x,abs(H_21_esti))
title('Canal h21')
subplot (2,2,4)
plot(x,abs(H_22_re),x,abs(H_22_esti))
title('Canal h22')
```

4) Representa3.m

```
function
[]=representa3(H_11_re,H_12_re,H_21_re,H_22_re,H_11_real_ML,H_11_imag_ML,H_12_real_ML,H_12_imag_ML,
H_21_real_ML,H_21_imag_ML,H_22_real_ML,H_22_imag_ML)

H_11_real=[zeros(28,1); H_11_real_ML(2089:2289); zeros(27,1)];
H_11_imag=[zeros(28,1); H_11_imag_ML(2089:2289); zeros(27,1)];
H_12_real=[zeros(28,1); H_12_real_ML(2089:2289); zeros(27,1)];
H_12_imag=[zeros(28,1); H_12_imag_ML(2089:2289); zeros(27,1)];
H_21_real=[zeros(28,1); H_21_real_ML(2089:2289); zeros(27,1)];
H_21_imag=[zeros(28,1); H_21_imag_ML(2089:2289); zeros(27,1)];
H_22_real=[zeros(28,1); H_22_real_ML(2089:2289); zeros(27,1)];
H_22_imag=[zeros(28,1); H_22_imag_ML(2089:2289); zeros(27,1)];

H_11_esti=H_11_real+H_11_imag*j;
H_12_esti=H_12_real+H_12_imag*j;
H_21_esti=H_21_real+H_21_imag*j;
H_22_esti=H_22_real+H_22_imag*j;

x=1:256;
subplot (2,2,1)
```

```
plot(x,abs(H_11_re),x,abs(H_11_esti))
title('Canal h11')
subplot (2,2,2)
plot(x,abs(H_12_re),x,abs(H_12_esti))
title('Canal h12')
subplot (2,2,3)
plot(x,abs(H_21_re),x,abs(H_21_esti))
title('Canal h21')
subplot (2,2,4)
plot(x,abs(H_22_re),x,abs(H_22_esti))
title('Canal h22')
```

5) Chan_SUI.m

```
%Cálculo de los paths del canal SUI-3 modificado.

function[paths]=chan_SUI(N,M,P,K,tau,Dop,Fnorm)

% Cálculo de la potencia en las componentes constante y aleatoria de la
% distribución Rice para cada tap.
P=10.^(P/10);      % Cálculo de la potencia lineal.
s2=P./(K+1);       % Cálculo de la varianza.
m2=P.*(K./(K+1));  % Cálculo de la componente constante de potencia.
m=sqrt(m2);        % Cálculo de la parte constante.

% Creación de los componentes de canal Ricean con la potencia especificada.

% Número de taps.
L=length(P);
% Parte aleatoria de los paths.
paths_r=sqrt(1/2)*(randn(L,N)+j*randn(L,N)).*((sqrt(s2))'*ones(1,N));
% Parte constante de los paths.
paths_c=m'*ones(1,N);

% Antes de la combinación del conjunto de coeficientes, se forma el
% espectro en función de la densidad de potencia de la función Doppler.
% La función 'FFTFILT' espera los coeficientes del filtro en el dominio
% temporal, luego se deben calcular. El filtro se normaliza en el dominio
% del tiempo.
for p=1:L
   % Normalización a la frecuencia máxima de Doppler.
   D=Dop(p)/max(Dop)/2;
   % Vector de frecuencias.
   f0=[0:M*D]/(M*D);
   % Aproximación de la PSD.
   PSD=0.785*f0.^4 - 1.72*f0.^2+1.0;
   % S(f).
   filt=[ PSD(1:end-1) zeros(1,M-2*M*D) PSD(end:-1:2) ];
   % Paso de S(f) a |H(f)|.
   filt=sqrt(filt);
   % Obtención de la respuesta al impulso.
   filt=ifftshift(ifft(filt));
   % Parte real de la respuesta impulsiva.
   filt=real(filt);
   % Normalización.
```

```
    filt=filt / sqrt(sum(filt.^2));
    path=fftfilt(filt, [paths_r(p,:) zeros(1,M)]);
    paths_r(p,:)=path(1+M/2:end-M/2);
end
paths=paths_r+paths_c;

% Aplicación del factor de normalización.
paths=paths*10^(Fnorm/20);
paths_r=paths_r*10^(Fnorm/20);
paths_c=paths_c*10^(Fnorm/20);
```

Anexo B

El objetivo de este anexo es proporcionar los códigos VHDL utilizados en el presente proyecto.

1) Cliclos.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


---- Uncomment the following library declaration if instantiating

---- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;


entity ciclos is

        port (ce, clk, h : in std_logic;

                cnt : out std_logic_vector(31 downto 0));

end ciclos;


architecture Behavioral of ciclos is


signal tmp : std_logic_vector (31 downto 0):=(others=>'0');


begin


process (clk, ce, h)
```

```
begin
        if h<='0' then
        tmp<=(others=>'0');
        elsif (ce='1' and rising_edge(clk)) then
                        tmp<=tmp+1;
        end if;

end process;
cnt<=tmp;
end Behavioral;
```

2) Control_mux.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity control_mux is
        port (tmp : in std_logic_vector(31 downto 0);
                        sel : out std_logic);
end control_mux;
```

```
architecture Behavioral of control_mux is


begin


process (tmp)


begin


if tmp<1472 then
        sel<='0';
else
        sel<='1';
end if;


end process;


end Behavioral;
```

3) Carga_lectura4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity carga_lectura4 is
        port (tmp : in std_logic_vector (31 downto 0);
                carga, lectura, mux : out std_logic);
end carga_lectura4;

architecture Behavioral of carga_lectura4 is

begin

process (tmp)

begin

        if tmp<256 then
                carga<='1';
                lectura<='0';
        elsif (tmp>=1168 and tmp<1424) then
                carga<='0';
                lectura<='1';
        elsif (tmp>=1472 and tmp<1728) then
                carga<='0';
                lectura<='1';
        else
                carga<='0';
                lectura<='0';
        end if;

        if tmp<1472 then
```

```
                    mux<='1';
        else
                    mux<='0';
        end if;
end process;


end Behavioral;
```

4) Contador3.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity contador3 is
        port (tmp : in std_logic_vector(31 downto 0);
                    ce, clk : in std_logic;
                    cnt : out std_logic_vector(7 downto 0));
end contador3;

architecture Behavioral of contador3 is

signal tmp_1 : std_logic_vector (7 downto 0):=(others=>'0');
```

```
begin

process (clk)

begin

        if (ce='1' and rising_edge(clk)) then
                if (tmp>=585 and tmp<841) then
                        tmp_1<=tmp_1+1;
                elsif (tmp>=2057 and tmp<2313) then
                        tmp_1<=tmp_1+1;
                else
                        tmp_1<=(others=>'0');
                end if;
        end if;
end process;
cnt<=tmp_1;
end Behavioral;
```

5) Captura3.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
```

```
--library UNISIM;

--use UNISIM.VComponents.all;


entity captura3 is

                port (entrada : in std_logic_vector (23 downto 0);

                                o_1, o_2 : out std_logic_vector (23 downto 0);

                                clk , ce , carga, lectura, mux : in std_logic);


end captura3;


architecture Behavioral of captura3 is


type regist is array (0 to 255) of std_logic_vector (23 downto 0);

signal senal_en: regist;

signal x : std_logic_vector(7 downto 0):=(others=>'0');



begin


  process(entrada, clk, carga, lectura, mux)

  begin

        if (ce ='1' and rising_edge(clk)) then

                        if carga ='1' then

                                senal_en(conv_integer(x)) <= entrada;

                                x <= x + 1;

                        elsif lectura='1' then

                                x <= x + 1;

                        end if;
```

```
                end if;


                if mux='1' then
                        if carga ='1' then
                                        o_1 <= (others => '0');
                                        o_2 <= (others => '0');
                        elsif lectura='1' then
                                        o_1 <= senal_en(conv_integer(x));
                                        o_2 <= (others => '0');
                        end if;
                else
                        if carga ='1' then
                                        o_2 <= (others => '0');
                                        o_1 <= (others => '0');
                        elsif lectura='1' then
                                        o_2 <= senal_en(conv_integer(x));
                                        o_1 <= (others => '0');
                        end if;
                end if;


        end process;


end Behavioral;
```

6) Pruebamult2.vhd

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;


entity pruebamult2 is

 port (

-- SEÑALES DE ENTRADA

   A : in std_logic_vector(23 downto 0);

   B : in std_logic_vector(23 downto 0);

   F : out std_logic_vector(23 downto 0)

   );

end pruebamult2;


architecture Behaivoural of pruebamult2 is


 signal a_int, b_int : signed(23 downto 0);

 signal f_int: signed(47 downto 0);

begin


--Conversión de tipos

 a_int <= signed(A);

 b_int <= signed(B);

 F <= std_logic_vector(f_int(37 downto 14));


--PROCESO COMBINACIONAL


 Mult2 : process (a_int,b_int)
```

```
 variable f_vari : signed(47 downto 0);
  begin
    f_vari:= a_int * b_int;
    f_int <= f_vari;
  end process;
end Behaivoural;
```

7) pruebarestador.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity pruebarestador is
  port (
-- SEÑALES DE ENTRADA
    A : in std_logic_vector(23 downto 0);
    B : in std_logic_vector(23 downto 0);
    F     : out std_logic_vector(23 downto 0)
    );
end pruebarestador;

architecture Behaivoural of pruebarestador is

  signal a_int, b_int, f_int: signed(23 downto 0);

begin

--Conversión de tipos
```

```
 a_int <= signed(A);
 b_int <= signed(B);
 F <= std_logic_vector(f_int);

--PROCESO COMBINACIONAL

 Sum : process (a_int,b_int)
variable f_vari : signed(23 downto 0);
 begin
  f_vari:= a_int - b_int;
  f_int <= f_vari;
 end process;
end Behaivoural;
```

8) pruebasumador.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity pruebasumandor is
 port (
-- SE ALES DEENTRADA
   A : in std_logic_vector(23 downto 0);
   B : in std_logic_vector(23 downto 0);
   F : out std_logic_vector(23 downto 0)
   );
end pruebasumandor;

architecture Behaivoural of pruebasumandor is
```

```
 signal a_int, b_int, f_int: signed(23 downto 0);


begin


--Conversión de tipos

 a_int <= signed(A);

 b_int <= signed(B);

 F <= std_logic_vector(f_int);


--PROCESO COMBINACIONAL


 Sum : process (a_int,b_int)

variable f_vari : signed(23 downto 0);

 begin

  f_vari:= a_int + b_int;

  f_int <= f_vari;

 end process;

end Behaivoural;
```

9) registro.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity registro is

                port (entrada : in std_logic_vector (23 downto 0);

                        o_1 : out std_logic_vector (23 downto 0);
```

```
                                        clk , ce : in std_logic);
end registro;

architecture Behavioral of registro is

signal x, y : std_logic_vector(23 downto 0);

begin

  process(clk, ce)
  begin
                        if (ce ='1' and rising_edge(clk)) then
                                        x<=entrada;
                                        y<=x;
                        end if;
            end process;
            o_1<=y;
end Behavioral;
```

10) registro1.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity registro is
                        port (entrada : in std_logic_vector (23 downto 0);
                                        o_1 : out std_logic_vector (23 downto 0);
                                        clk , ce : in std_logic);
```

```
end registro;


architecture Behavioral of registro is


signal x : std_logic_vector(23 downto 0);


begin


  process(clk, ce)

  begin

                        if (ce ='1' and rising_edge(clk)) then

                                                x<=entrada;

                        end if;

              end process;

              o_1<=x;

end Behavioral;
```

11) desplazamiento.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity desplazamiento is

                        port (entrada : in std_logic_vector (23 downto 0);

                                                o_1 : out std_logic_vector (23 downto 0));

end desplazamiento ;


architecture Behavioral of desplazamiento  is
```

```
        signal a_int : signed(23 downto 0);

begin

        a_int <= signed(entrada);

        process(a_int)
        begin
                if a_int<0 then
                        o_1<='1'&entrada(23 downto 1);
                else
                        o_1<='0'&entrada(23 downto 1);
                end if;
        end process;

end Behavioral;
```

12) Pruebasumador2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity pruebasumandor is
 port (
-- SEÑALES DE ENTRADA
   A : in std_logic_vector(23 downto 0);
   B : in std_logic_vector(23 downto 0);
   F : out std_logic_vector(23 downto 0)
```

```
  );
end pruebasumandor;

architecture Behaivoural of pruebasumandor is

 signal a_int, b_int, f_int: signed(23 downto 0);

begin

--Conversión de tipos
 a_int <= signed(A);
 b_int <= signed(B);
 F <= std_logic_vector(f_int);

--PROCESO COMBINACIONAL

 Sum : process (a_int,b_int)
variable f_vari : signed(23 downto 0);
 begin
   f_vari:= a_int + b_int;
   f_int <= f_vari;
 end process;
end Behaivoural;
```

13) habilitacion.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


---- Uncomment the following library declaration if instantiating

---- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;


entity habilitacion is

        port (tmp : in std_logic_vector (31 downto 0);

                        en : out std_logic);

end habilitacion;


architecture Behavioral of habilitacion is


begin


process (tmp)


begin


        if (tmp>=583 and tmp<584) then

                        en<='1';

        else

                        en<='0';
```

```
            end if;
end process;


end Behavioral;
```

14) multiplexor.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity multiplexor is
   Port ( a : out  STD_LOGIC_VECTOR (23 downto 0);
                                    b : out  STD_LOGIC_VECTOR (23 downto 0);
                                    lectura, escritura, ce, clk : in  STD_LOGIC;
          c : in  STD_LOGIC_VECTOR (23 downto 0));
end multiplexor;

architecture Behavioral of multiplexor is

type regist is array (0 to 127) of std_logic_vector (23 downto 0);
signal senal_en: regist;
signal x : std_logic_vector(6 downto 0):=(others=>'0');
```

```
begin


process(c,ce,lectura, escritura,clk)


begin


if (ce ='1' and rising_edge(clk)) then
                                        if escritura ='1' then
                                                senal_en(conv_integer(x)) <= c;
                                                x<=x+1;
                                        elsif lectura='1' then
                                                x<=x+1;
                                        end if;
end if;


if escritura='1' then
        a<=(others=>'0');
        b<=(others=>'0');
elsif lectura='1' then
        b<=c;
        a<=senal_en(conv_integer(x));
else
        a<=(others=>'0');
        b<=(others=>'0');
end if;


end process;
```

```
end Behavioral;
```

15) Pruebamult22.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity pruebamult22 is
 port (
-- SEÑALES DE ENTRADA
   A : in std_logic_vector(23 downto 0);
   B : in std_logic_vector(23 downto 0);
   F : out std_logic_vector(31 downto 0)
   );
end pruebamult22;

architecture Behaivoural of pruebamult22 is

 signal a_int, b_int : signed(23 downto 0);
 signal f_int: signed(47 downto 0);
begin

--Conversión de tipos
 a_int <= signed(A);
 b_int <= signed(B);
 F <= std_logic_vector(f_int(46 downto 15));
```

```
--PROCESO COMBINACIONAL


 Mult2 : process (a_int,b_int)
variable f_vari : signed(47 downto 0);
 begin
  f_vari:= a_int * b_int;
  f_int <= f_vari;
 end process;
end Behaivoural;
```

16) Pruebasumador22.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;


entity pruebasumandor22 is
 port (
-- SE ALES DEENTRADA
   A : in std_logic_vector(31 downto 0);
   B : in std_logic_vector(31 downto 0);
   F : out std_logic_vector(31 downto 0)
   );
end pruebasumandor22;


architecture Behaivoural of pruebasumandor22 is


 signal a_int, b_int, f_int: signed(31 downto 0);


begin
```

```
--Conversión de tipos
  a_int <= signed(A);
  b_int <= signed(B);
  F <= std_logic_vector(f_int);

--PROCESO COMBINACIONAL

  Sum : process (a_int,b_int)
 variable f_vari : signed(31 downto 0);
  begin
    f_vari:= a_int + b_int;
    f_int <= f_vari;
  end process;
end Behaivoural;
```

17) Pruebarestador22.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity pruebarestador22 is
  port (
-- SE ALES DEENTRADA
    A : in std_logic_vector(31 downto 0);
    B : in std_logic_vector(31 downto 0);
    F     : out std_logic_vector(31 downto 0)
    );
end pruebarestador22;
```

```
architecture Behaivoural of pruebarestador22 is

 signal a_int, b_int, f_int: signed(31 downto 0);

begin

--Conversión de tipos
 a_int <= signed(A);
 b_int <= signed(B);
 F <= std_logic_vector(f_int);

--PROCESO COMBINACIONAL

 Sum : process (a_int,b_int)
variable f_vari : signed(31 downto 0);
 begin
  f_vari:= a_int - b_int;
  f_int <= f_vari;
 end process;
end Behaivoural;
```

18) Carga_lectura3.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
```

```
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity carga_lectura3 is
        port (tmp : in std_logic_vector (31 downto 0);
                        carga, lectura : out std_logic);
end carga_lectura3;

architecture Behavioral of carga_lectura3 is

begin

process (tmp)

begin

        if (tmp>=1446 and tmp<1447) then
                        carga<='1';
                        lectura<='0';
        elsif (tmp>=1447 and tmp<1703) then
                        carga<='0';
                        lectura<='1';
        else
                        carga<='0';
                        lectura<='0';
        end if;
end process;
```

```
end Behavioral;
```

19) correcion.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


---- Uncomment the following library declaration if instantiating

---- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;


entity correccion is

                port (entrada : in std_logic_vector (23 downto 0);

                                o_1 : out std_logic_vector (28 downto 0);

                                clk , ce , carga, lectura : in std_logic);


end correccion;


architecture Behavioral of correccion is


signal x : signed(23 downto 0);

signal i : signed(8 downto 0):=(others=>'0');

signal o_sig : signed(32 downto 0);


begin
```

```
process(entrada, clk, carga, lectura)
begin
        if (ce ='1' and rising_edge(clk)) then
                if carga ='1' then
                        x <= signed(entrada);
                elsif lectura='1' then
                        i <= i + 1;
                        if i=255 then
                                i<=(others=>'0');
                        end if;
                end if;
        end if;

        if carga ='1' then
                        o_sig <= (others => '0');
        elsif lectura='1' then
                        o_sig <= i*x;
        else
                        o_sig<=(others=>'0');
        end if;

    end process;
o_1<=std_logic_vector(o_sig(28 downto 0));

end Behavioral;
```

Referencias

[1] *Ogawa Y.; Nishio K.; Mishimura T. and Ohgane T. Channel and frequency offset estimation for a MIMO-OFDM system. IEEE 60th Vehicular Technology Conference. VTC2004-Fall. 26-29 Sept. 2004. Page(s): 1523 - 1527 Vol. 2.*

[2] *M. Julia Fernández-Getino García; Ezio Biglieri and Giorgio Taricco. Frequency-Domain Channel Estimation in MIMO-OFDM. The 12th European Signal Processing Conference (EUSIPCO). Viena (Austria), pp1869-1872, Sept.2004.*

[3] *Tae-Hwan Kim and In-Cheol Park. Two-Step appproah for Coarse Time Synchronization and Frequency Offset Estimation For IEEE 802.16d Systems. Department of Electrical Engineering. Korea Advanced Institute of Science and Technology (KAIST) 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea.*

[4] *System Generator For DSP. Reference Guide. Release 10.1 March, 2008*

[5] *http://www.coit.es (Último acceso Octubre 2010)*

[6] *Ye G. Li; M. Seshadri and S. Ariyavisitakul. Channel estimation for OFDM systems with transmitter diversity in mobile wireless channels. IEEE Journal on Selected areas in Communications. Vol.17, no3, pag: 461-471. March 1999.*

[7] *Han Zhang; Jiming Chen; Youxi Tang and Shaoqian Li. Analysis of pilot-symbol aided channel estimation for MIMO-OFDM systems. International Conference on Communications, Circuits and Systems. ICCCAS. 27-29 June 2004 Page(s):299 – 303 Vol.1.*

[8] *Sumei Sun; Wiemer I.; Ho C.K. and Tjhung T.T. Training sequence assisted channel estimation for MIMO-OFDM. IEEE Wireless Communications and Networking. WCNC, 16-20 March 2003. Page(s):38 - 43 vol.1.*

[9] *Changho Suh; Chan-Soo Hwang and Hokyu Choi. Preamble design for channel estimation in MIMO-OFDM systems. IEEE Global Telecommunications Conference. GLOBECOM '03. 1-5 Dec. 2003. Page(s):317 - 321 Vol.1.*

[10] *Changho Suh; Chan-Soo Hwang and Hokyu Choi. Comparative study of time-domain and frequency-domain channel estimation in MIMOOFDM systems. 14th IEEE Proceedings on Personal, Indoor and Mobile Radio Communications. PIMRC, 7-10 Sept. 2003 Page(s):1095 - 1099 vol.2.*

[11] *Wang Dongming; Han Bing; Zhao Junhui; Gao Xiqi and You Xiaohu. Channel estimation algorithms for broadband MIMO-OFDM sparse channel. 14th IEEE Proceedings on Personal, Indoor and Mobile Radio Communications. PIMRC. 7-10 Sept. 2003 Page(s):1929 - 1933 vol.2.*

[12] *Rahman Q.M. and Hefnawi M. Channel estimation methods for MIMO-OFDM system: time domain versus frequency domain. Canadian Conference on Electrical and Computer Engineering. 2-5 May 2004 Page(s):689 - 692 Vol.2.*

[13] *Han Zhang; Jiming Chen; Youxi Tang and Shaoqian Li. Analysis of pilot-symbol aided channel estimation for MIMO-OFDM systems. International Conference on Communications, Circuits and Systems. ICCCAS. 27-29 June 2004. Page(s):299 – 303 Vol.1.*

[14] *Zhongshan Wu; Jianqiang He and Guoxiang Gu. Design of optimal pilot-tones for channel estimation in MIMO-OFDM systems. IEEE Wireless Communications and Networking Conference. 13-17 March 2005. Page(s):12 - 17 Vol. 1.*

[15] *Bai W.; He C.; Jiang L.G. and Li X.X. Robust channel estimation in MIMO-OFDM systems. Electronics Letters Volume 39. 23 Jan 2003. Page(s):242 - 244.*

[16] *Ercer V.; Hari K.V.S.; Smith M.S. and Baum D.S. et al. Channel models for fixed wireless applications. Contribution IEEE 802.16.3c- 01/29r1, Feb 2001.*

[17] *http://www.mathworks.com/products/matlab/ (Último acceso Octubre 2010)*

[18] *Javier García de Jalón, José Ignacio Rodríguez, Jesús Vidal. Manual aprenda Matlab 7.0 como si estuviera en primero. Madrid, diciembre 2005.*

[19] *Capitulo2. Evaluación de herramientas de alto nivel para diseño hardware. Manuel Gutiérrez Rizo. I. Telecomunicación.*

[20] *http://www.mathworks.com/products/simulink/ (Último acceso Octubre 2010)*

[21] *http://www.jcelectronica.com/articles/FPGA%20and%20MCU.htm (Último acceso Octubre 2010)*

[22] *System Generator For DSP. Reference Guide. Release 10.1 March, 2008*

[23] *Capitulo3. Evaluación de herramientas de alto nivel para diseño hardware. Manuel Gutiérrez Rizo. I. Telecomunicación.*

[24] *http://www.lyrtech.com/ (Último acceso Octubre 2010)*

[25] *http://sedici.unlp.edu.ar/ARG-UNLP-TDG-0000000022/94.pdf (Último acceso Octubre 2010)*

[26] *Moose P.H. A technique for Orthogonal Frequency Division Multiplexing Frequency Offset Correction. IEEE Transactions on Communications. Vol 42, pp. 2908-2914. October 1994.*

[27] *Jongmin Cho, Youngmin Cho, Mohammad Rakibul Islam, Jinsang Kim, and Won-Kyung Cho . Hardware-Efficient Auto-Correlation for Synchronization of MIMO-OFDM WLAN Systems. Computer System Architecture & VLSI Laboratory. Department of Electronics and Radio Engineering, Kyung Hee University. 1 Seocheon, Kihung, Yongin, Gyunggi, 446-701, Republic of Korea*

[28] *Low-Power and High-Accurate Synchronization for IEEE 802.16d Systems. Tae-Hwan Kim, Student Member, IEEE, and In-Cheol Park, Senior Member, IEEE*

[29] *van de Beek J.J.; Sandell M.; Isaksson M. and Börjesson P. Lowcomplex frame synchronization in OFDM systems. Proc. ICUPC, pp.982-986, Nov. 6-10, 1995.*

[30] *Classen F. and Meyr H. Synchronization algorithms for an OFDM system for mobile communication. ITG- Fachtagung, pp 105-113, Oct. 26-28, 1994.*

[31] *Timothy M. Schmidl and Donald C. Cox. Robust Frequency and Timing Synchronization for OFDM. IEEE Transaction on Communications. Vol 45, no.12 December 1997.*

[32] *Park B.; Cheon H.; Kang C. and Hong D. A novel timing estimation method for OFDM systems. IEEE Communications Letters, vol. 7, No. 5, May 2003.*

[33] *Jose Angel Rivas Cantero. M. Julia Femaindez-Getino Garcia. Channel Estimation and Frequency Synchronization for a Multi-Antenna Wimax System. Dpto. de Teoria de la Seal y Comunicaciones. Universidad Carlos III de Madrid.*

[34] *Jose Angel Rivas Cantero. Estimación de canal y sincronización de frecuencia en un sistema MIMO-OFDM compatible con el estándar wimax IEEE 802.16-2004. Universidad Carlos III de Madrid. Junio 2006.*

[35] *David Díaz Martín y Roberto Prieto Alonso. Estudio práctico sobre el prototipado de un sistema MIMO 2x2 para WiMAX. Estudio Tecnológico de la Universidad Carlos III de Madrid. Diciembre 2007.*

[36] *David Díaz Martín. Prototipado de un sistema wimax MIMO 2x2. Proyecto Fin de Carrera Universidad Carlos III de Madrid. Enero 2010.*

[37] *Bölcskey H. and Paulraj A.J. Multiple Input - Multiple Output Wireless System. The Communications Handbook, 2oed, CRC Press, 2002.*

[38] *IEEE Standard for local and metropolitan area networks part 16: Air interface for fixed broadband wireless access systems IEEE Std 802.16-2004.*